

Multimodal Long-Term Video Understanding

Medhini Gulganjalli Narasimhan



Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-206

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-206.html>

August 10, 2023

Copyright © 2023, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Multimodal Long-Term Video Understanding

By

Medhini Narasimhan

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Trevor Darrell, Chair

Professor Alexei Efros

Professor Jitendra Malik

Professor Cordelia Schmid

Summer 2023

Multimodal Long-Term Video Understanding

Copyright 2023
by
Medhini Narasimhan

Abstract

Multimodal Long-Term Video Understanding

by

Medhini Narasimhan

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Trevor Darrell, Chair

The internet hosts an immense reservoir of videos, witnessing a constant influx of thousands of uploads to platforms like YouTube every second. These videos represent a valuable repository of multimodal information, providing an invaluable resource for understanding audio-visual-text relationships. Moreover, understanding the content in *long* videos (think 2 hours), is an open problem. This thesis investigates the intricate interplay between diverse modalities—audio, visual, and textual—in videos and harnesses their potential for comprehending semantic nuances within long videos. My research explores diverse strategies for combining information from these modalities, leading to significant advancements in video summarization and instructional video analysis.

The first part introduces an approach to synthesizing long video textures from short clips by rearranging segments coherently, while also considering audio conditioning. The second part discusses a novel technique for generating concise visual summaries of lengthy videos guided by natural language cues. Additionally, we focus specifically on summarizing instructional videos, capitalizing on audio-visual alignments and task structures to produce informative summaries. To further enrich the comprehension of instructional videos, the thesis introduces a cutting-edge approach that facilitates the learning and verification of procedural steps within instructional content, empowering the model to grasp long and complex video sequences and ensure procedural accuracy. Lastly, the potential of large language models is explored for answering questions about images through code generation. Through comprehensive experiments, the research demonstrates the efficacy of the proposed methodologies, envisioning promising future prospects in the field of semantics in long videos by integrating audio, visual, and textual relationships.

To my beloved mom Latha, for everything I am today.

Contents

Contents	ii
1 Introduction	1
2 Audio-Conditioned Contrastive Video Textures	5
2.1 Introduction	5
2.2 Related Work	8
2.3 Contrastive Video Textures	9
2.4 Audio-Conditioned Video Textures	11
2.5 Experiments	12
2.6 Conclusion	16
3 CLIP-It! Language-Guided Video Summarization	19
3.1 Introduction	19
3.2 Related Work	21
3.3 CLIP-It: Language-Guided Video Summarization	22
3.4 Experiments	25
3.5 Discussion and Broader Impacts	33
4 TL;DW? Summarizing Instructional Videos with Task Relevance & Cross-Modal Saliency	34
4.1 Introduction	35
4.2 Related Work	36
4.3 Summarizing Instructional Videos	37
4.4 Instructional Video Summarization Datasets	40
4.5 Experiments	42
4.6 Conclusion	47
5 Learning and Verification of Task Structure in Instructional Videos	48
5.1 Introduction	48
5.2 Related Works	50
5.3 Learning Task Structure through Masked Modeling of Steps	50

5.4	Datasets and Evaluation Metrics	55
5.5	Experiments	56
5.6	Conclusion	60
6	Modular Visual Question Answering via Code Generation	62
6.1	Introduction	62
6.2	Related Work	63
6.3	Few-shot VQA via Code Generation	64
6.4	Experiments	66
6.5	Conclusion	69
6.6	Limitations	70
6.7	Acknowledgements	70
7	Conclusion	71
	Bibliography	74
A	Chapter 2 Supplementary Material	87
A.1	Implementation Details	87
A.2	Unconditional Video Textures: Baselines	88
A.3	Video Quality Metric	89
A.4	Unconditional Contrastive Audio-Video Textures	90
A.5	Comparing Transition Probabilities	90
B	Chapter 3 Supplementary Material	93
B.1	Implementation Details	93
B.2	Additional Results	94
B.3	Limitations	95
C	Chapter 4 Supplementary Material	96
C.1	<i>WikiHow Summaries</i> Data Collection	96
C.2	Implementation Details	97
C.3	Additional Results	98
C.4	Additional Quantitative Results	101
D	Chapter 5 Supplementary Material	103
D.1	Implementation Details	103
D.2	Additional Quantitative Results	104
D.3	Additional Qualitative Results	104
E	Chapter 6 Supplementary Material	108
E.1	GradCAM	108
E.2	Implementation Details	108

E.3	Details on Baselines	109
E.4	Qualitative Comparisons	110
E.5	Additional Quantitative Results	113
E.6	Experiments with Additional Primitives	113
E.7	Licenses and Other Dataset Details	115

Acknowledgments

My Ph.D. journey wouldn't have been possible without the unwavering support and guidance of some incredible people. Below is my humble attempt to put into words, my gratitude for all those who have played a role in this accomplishment.

I'd like to thank all the members of my committee – Trevor Darrell, Alexei Efros, Jitendra Malik, and Cordelia Schmid. Their valuable insights through my Ph.D. journey have helped this thesis take shape and form.

I'm especially grateful to my advisor, Trevor, for taking me on as his student and mentoring me over the course of the last four years. From brainstorming ideas to paper writing, I've learned a lot from Trevor and grown in every aspect as a researcher. His ability to manage people and get a project going is akin to none, and I hope to lead in the same way someday. I've also greatly valued the freedom to work on any problem I aspired to throughout the Ph.D. Outside of research, Trevor has been extremely understanding about taking time off from work when needed, which I deeply appreciate. The pandemic hit 6 months into the program, which affected my work and I wouldn't have been able to push through without the encouragement and support from Trevor.

I feel privileged to have worked with Alyosha and it's an experience I will celebrate for life. Of all the things I learned from his mentorship, the one that I'm most strongly influenced by is his romantic/artistic view of research. It's a philosophy that has given me a new-found respect for the field and the problems we solve day to day. I'm thankful for all of the outdoor hikes, camping trips, and adventures Alyosha organized, which helped me keep sane during stressful times.

I'm thankful to Cordelia for her mentorship during my year at Google and look forward to continuing working with her while I'm a full-time employee there. I'm very grateful to Jitendra for his feedback and constructive criticism of my research. I've learned a lot from his insightful discussions at meetings and his ability to recollect any vision paper ever written

I would've never in my wildest dreams dreamt of graduating from Berkeley and the real reason I got to Berkeley was the mentorship from my Master's advisors - Alex Schwing and Lana Lazebnik. I'm thankful to them for teaching me the As and Bs of research and for being patient with me as I struggled and for believing in my abilities when I didn't. I'm deeply indebted to Alex, who till today believes in my abilities more than I do. From analyzing every loss graph and experiment, to burning the midnight oil on overleaf at every paper deadline with me, Alex's enthusiasm and mentorship transformed me from a mere code monkey into a researcher. I'm constantly surprised by the time and effort Alex would invest in each of his students (or magicians, as Alex calls us), and I feel extremely fortunate to have been a part of his group.

Lana taught me the importance of dotting every i and crossing every t, whether it comes to writing a paper, making a figure, or presenting a poster. I significantly improved my writing, teaching, and overall research skills under her guidance. Lana's vision and research intuition has helped me stay grounded and true to the facts in every project I do, a quality I will carry with me for a lifetime.

This thesis and my research wouldn't be complete without the hard work of all of my collaborators. I'm extremely thankful for Anna Rohrbach, Andrew Owens, Arsha Nagrani, Miki Rubinstein, Chen Sun, Sanjay Subramanian, and Andy Zeng. Anna has been a part of every project I've worked

on at Berkeley, proofreading my work and investigating every aspect of it that would irk a reviewer. I'm thankful for Arsha's mentorship during my internship at Google, and for her zeal which got me through a lot of lows of pandemic research.

I've learned a ton from my peers at Berkeley and for that I'm forever grateful to everyone in BAIR and the Vision lab. I'd like to thank Aleks, Alvin, Amir, Angjoo, Antonio, Ashish, Baifeng, Boyi, Devin, Ethan, Grace, Harry, Ilija, Jathushan, Kartik, Lisa, Norman, Neerja, Parsa, Roei, Rudy, Sasha, Sanjay, Shubham, Seth, Suzie, Tete, Vongani, Vickie and all the other members of the vision lab. All the board game socials, lab ski, and camping trips, and the hikes in Berkeley hills made this experience a lot more cherishable. Special thanks to Colorado Reed and Ritwik Gupta who co-led Berkeley Climate Initiative with me. Thankful to the BAIR and EECS Admin team - Angie, Ami, Roxana, Lena, Jean, and Shirley for making it seamless to deal with logistics.

I'm thankful for all the love I've received from my family. To my mom, thank you for making me the person I am today. Thank you for being there by my side each day, for riding the highs and lows of life from the start, and especially during my Ph.D. My mom is the biggest source of inspiration in my life, and I aspire to be half as bold, adventurous, and industrious as she is. She defined what hard work and success looks like very early on during my childhood, which shaped me into a relentless, daring individual with big dreams. I'm forever grateful to my granddad, who was the first and best engineer/scientist I knew. With three patents in his sixties, his wit and knack for tinkering with any device he could get his hands on piqued curiosity in me. My dad taught me to be meticulous in my work and all life habits which has helped me cope with tough times. I'm thankful to my aunt and uncle, Kalai and Giri, who were my first research mentors and have since played a pivotal role in my life, research and outside. I'm thankful for the fuzzleballs in my life - my pets Nano, Chilli, and Ginger.

To my friends and my support system - what would I do without you! I've been blessed with some amazing friends who have cheered me on rainy days when I needed it the most. I'm thankful to my incredible roommate and friend, Suzie, for talking me up on my dull days, partaking in every crazy hike/trip/hangout I've organized, and inspiring me to be a better researcher and person. To my friend Anirudh, thanks for being my closest friend and companion since high school. To friends who are just a phone call away - Corentin, Divy, and Suddhu, thank you for picking up and always being there! A big thank you to all my cousins and friends - Supreetha, Shvenak, Rahul, Archana, Kartik, Justin Kerr, Ale, Ethan, Daniel, Fred, Matilde, Scott, Zhang, Kristen, Niladri, GT, Kishor, Neerja, Justin Wong, Kevin, Andrew, Unnat, Daniel and anyone else I may have forgotten.

I've always enjoyed reading the acknowledgment section of every thesis, so if you made it this far, thank you for reading!

Chapter 1

Introduction

Our comprehension of commonplace actions, such as throwing, catching, and eating, is intrinsically linked to motion. The dynamic interplay and object interactions captured in videos play a pivotal role in shaping our perception of the surrounding world. Additionally, audio within videos provides crucial auditory cues, enriching our understanding of how objects sound and interact with one another. For instance, the sound of a ball bouncing or the clinking of cutlery while eating informs our cognitive processing. The combination of speech transcribed as text, sound, video, and time (chronological arrangement of events within the video) renders them multimodal and multifaceted sources of information.

The intrinsic correlations among different modalities in videos enable the prediction of one modality from another, as evident in the predictability of speech from audio and the temporal correspondence of visuals. Leveraging these relationships, training a network to predict one modality from another enables learning multimodal video representations in a self-supervised fashion. Several research works have explored this approach [106, 147, 103, 178, 177]. Notably, VideoBERT [147] pioneered the adaptation of the powerful BERT model to learn a joint visual-linguistic representation for video through masking. Additionally, MIL-NCE [103] exploits the alignment between transcripts and visuals in instructional videos, while Merlot [178] learns multimodal script knowledge through contrastive frame-caption matching and temporal ordering objectives.

These seminal works have thoroughly evaluated the learned representations through downstream action recognition and localization tasks, showcasing their efficacy. However, the emphasis in existing action recognition research predominantly centers on short video clips, primarily focusing on atomic actions. Consequently, a critical gap remains in understanding long videos, where we must interpret more than just isolated actions. For instance, consider an instructional video on making a pancake. Such a video comprises a sequence of atomic actions (*e.g.* “pick-up spoon”, “pour milk”), steps (*e.g.* “gather all ingredients”, “make pancake batter”), and events (*e.g.* “making a pancake”). To effectively comprehend such long videos, one must grasp the temporal context and the interplay between different elements. This challenge of understanding long videos constitutes an open problem in the field. Understanding, in this context, encompasses the comprehensive perception and interpretation of the underlying meaning and structure within the video content.

Another illustrative example of long videos can be found in movies, where the unfolding of

events necessitates grasping the temporal context to discern the plot progression and character development. In addressing this gap, video summarization emerges as a means to distill relevant and essential information from lengthy videos, enabling their effective breakdown into concise and informative representations. This process facilitates the extraction of critical content, contributing to a more comprehensive understanding of the long video’s content and structure.

My research focuses on designing models for understanding long videos and extracting semantics and structure from them. In this thesis, I present a comprehensive exploration of synthesis and summarization of long videos, with a focus on leveraging audio, visual, and language modalities. I discuss in detail our work on synthesizing long video textures followed by my works on video summarization – starting with language-guided video summarization and then more specifically summarizing the important steps in instructional videos. Focusing further on learning steps, I discuss my works on learning to identify mistakes in steps of instructional videos and using a large language model to break questions about an image into steps.

Audio-Conditioned Contrastive Video Textures In Chapter 2, we introduce a non-parametric approach for infinite video texture synthesis using a representation learned via contrastive learning. We take inspiration from Video Textures [132], which showed that plausible new videos could be generated from a single one by stitching its frames together in a novel yet consistent order. This classic work, however, was constrained by its use of hand-designed distance metrics, limiting its use to simple, repetitive videos. We draw on recent techniques from self-supervised learning to learn this distance metric, allowing us to compare frames in a manner that scales to more challenging dynamics, and to condition on other data, such as audio. We learn representations for video frames and frame-to-frame transition probabilities by fitting a video-specific model trained using contrastive learning. To synthesize a texture, we randomly sample frames with high transition probabilities to generate diverse temporally smooth videos with novel sequences and transitions. The model naturally extends to an audio-conditioned setting without requiring any finetuning. Our model outperforms baselines on human perceptual scores, can handle a diverse range of input videos, and can combine semantic and audio-visual cues in order to synthesize videos that synchronize well with an audio signal.

Language-Guided Video Summarization Chapter 3, introduces an approach to create visual summaries of long videos. A generic video summary is an abridged version of a video that conveys the whole story and features the most important scenes. Yet the importance of scenes in a video is often subjective, and users should have the option of customizing the summary by using natural language to specify what is important to them. Further, existing models for fully automatic generic summarization have not exploited available language models, which can serve as an effective prior for saliency. This chapter introduces *CLIP-It*, a single framework for addressing both generic and query-focused video summarization, typically approached separately in the literature. We propose a *language-guided* multimodal transformer that learns to score frames in a video based on their importance relative to one another and their correlation with a user-defined query (for query-focused summarization) or an automatically generated dense video caption (for generic

video summarization). Our model can be extended to the unsupervised setting by training without ground-truth supervision. We outperform baselines and prior work by a significant margin on both standard video summarization datasets (TVSum and SumMe) and a query-focused video summarization dataset (QFVS). Particularly, we achieve large improvements in the transfer setting, attesting to our method’s strong generalization capabilities.

Summarization Instructional Videos using Cross-Modal Similarity and Task Relevance

YouTube users looking for instructions for a specific task may spend a long time browsing content trying to find the right video that matches their needs. In Chapter 4, we focus on summarizing *instructional* videos, an under-explored area of video summarization. In comparison to generic videos, instructional videos can be parsed into semantically meaningful segments that correspond to important steps of the demonstrated task. Existing video summarization datasets rely on manual frame-level annotations, making them subjective and limited in size. To overcome this, we first automatically generate *pseudo summaries* for a corpus of instructional videos by exploiting two key assumptions: (i) relevant steps are likely to appear in multiple videos of the same task (*Task Relevance*), and (ii) they are more likely to be described by the demonstrator verbally (*Cross-Modal Saliency*). We propose an instructional video summarization network that combines a context-aware temporal video encoder and a segment-scoring transformer. Using pseudo summaries as weak supervision, our network constructs a visual summary for an instructional video given only video and transcribed speech. To evaluate our model, we collect a high-quality test set, *WikiHow Summaries*, by scraping WikiHow articles that contain video demonstrations and visual depictions of steps allowing us to obtain the ground-truth summaries. We outperform several baselines and a state-of-the-art video summarization model on this new benchmark.

Learning Task Structure from Instructional Videos Given the enormous number of instructional videos available online, learning a diverse array of multi-step task models from videos is an appealing goal. In Chapter 5, we introduce a new pre-trained video model, VideoTaskformer, focused on representing the semantics and structure of instructional videos. We pre-train VideoTaskformer using a simple and effective objective: predicting weakly supervised textual labels for steps that are randomly masked out from an instructional video (masked step modeling). Compared to prior work which learns step representations locally, our approach involves learning them globally, leveraging video of the entire surrounding task as context. From these learned representations, we can verify if an unseen video correctly executes a given task, as well as forecast which steps are likely to be taken after a given step. We introduce two new benchmarks for detecting mistakes in instructional videos, to verify if there is an anomalous step and if steps are executed in the right order. We also introduce a long-term forecasting benchmark, where the goal is to predict long-range future steps from a given step. Our method outperforms previous baselines on these tasks, and we believe the tasks will be a valuable way for the community to measure the quality of step representations. Additionally, we evaluate VideoTaskformer on 3 existing benchmarks—procedural activity recognition, step classification, and step forecasting—and demonstrate on each that our method outperforms existing baselines and achieves new state-of-the-art performance.

Modular Visual Question Answering via Code Generation In Chapter 6, we present a framework that formulates visual question answering as modular code generation. In contrast to prior work on modular approaches to VQA, our approach requires no additional training and relies on pre-trained language models (LMs), visual models pre-trained on image-caption pairs, and fifty VQA examples used for in-context learning. The generated Python programs invoke and compose the outputs of the visual models using arithmetic and conditional logic. Our approach improves accuracy on the COVR dataset by at least 3% and on the GQA dataset by 2% compared to the few-shot baseline that does not employ code generation.

This thesis delves into various challenges associated with decomposing and learning representations of long videos, resulting in substantial performance enhancements across video summarization and mistake detection benchmarks. Moreover, in light of the proliferation of large language models, the last work explores how the language capabilities of such models can be leveraged to benefit vision-language tasks. The final Chapter 7 critically examines the existing long-term video understanding models, identifying their limitations, and proposes potential solutions and future research directions based on the insights gleaned from this thesis.

Chapter 2

Audio-Conditioned Contrastive Video Textures

2.1 Introduction

We revisit Video Textures [132], a classic non-parametric video synthesis method which converts a single input video into an infinitely long and continuously varying video sequence. Video textures have been used to create dynamic backdrops for special effects and games, 3D portraits, dynamic scenes on web pages, and the interactive control of video-based animation [131, 130]. In these models, a new plausible video texture is generated by stitching together snippets of an existing video. Classic video texture methods have been very successful on simple videos with a high degree of regularity, such as a swinging pendulum. However, their reliance on Euclidean pixel distance as a similarity metric between frames makes them brittle to irregularities and chaotic movements, such as dances or performance of a musical instrument. They are also sensitive to subtle changes in brightness and often produce jarring transitions.

Representation-learning methods have made significant advances in the past decade and offer a potential solution to the limitations of classic video texture approaches. A natural approach may be to use Generative Adversarial Networks (GANs) [43] and/or Variational Autoencoders (VAEs) [73] which have achieved great success in generating images “from scratch”. Yet while video generation [82, 100, 154, 158, 161, 162] has shown some success, videos produced using such methods are unable to match the realism of actual videos. Current generative video methods fail to capture the typical temporal dynamics of real video and as a result fail on our task of synthesizing long and diverse video sequences conditioned on a single source video. In this work, we investigate contrastive learning [19, 20, 18] approaches to graph-based sequence generation,

This chapter is based on joint work with Andrew Owens, Alexei Efros, and Trevor Darrell and is presented much as it appeared in the WACV 2022, proceedings.

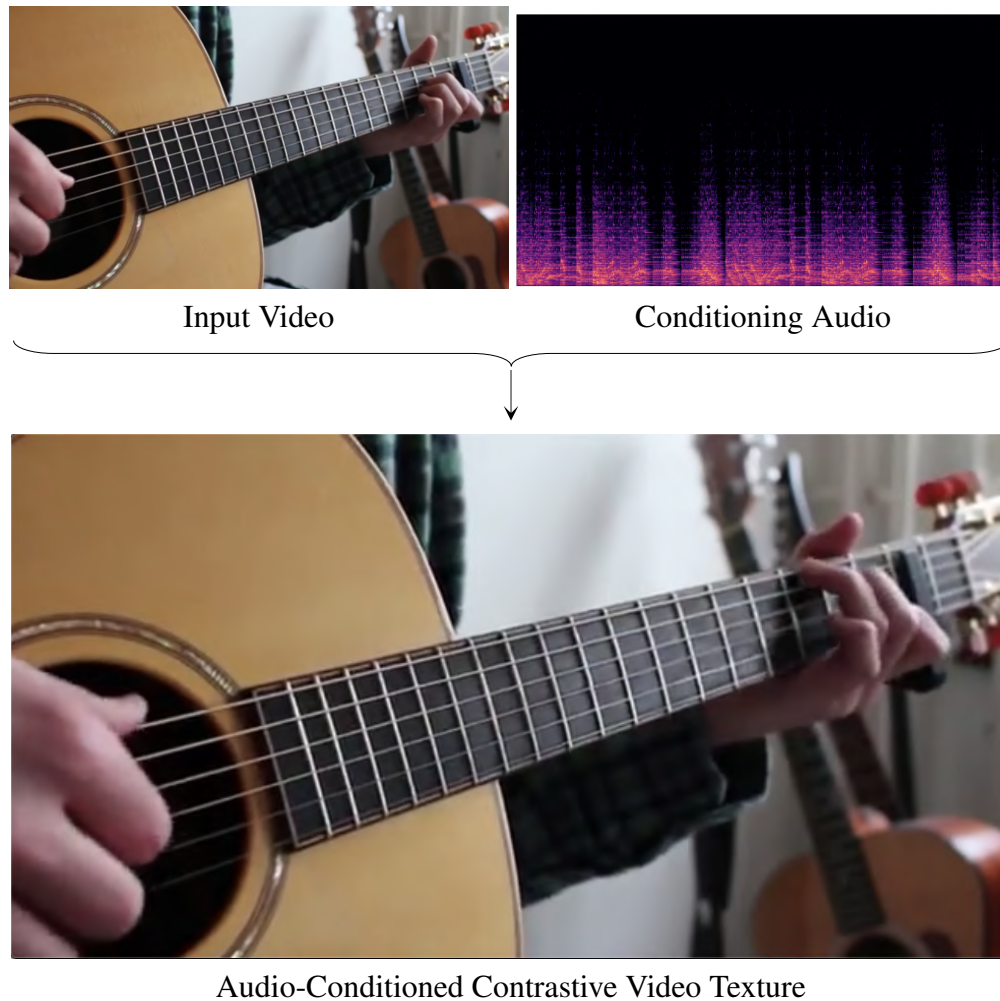


Figure 2.1: **Strumming to the Beat.** Click on each image to play the video/audio. We introduce Contrastive Video Textures, a learning-based approach for video texture synthesis. Given an input video and a conditioning audio, we extend our Contrastive model to synthesize a video texture that matches the conditioning audio.

conditional and unconditional, and demonstrate the ability of learned visual texture representations to render compelling video textures.

We propose Contrastive Video Textures, a non-parametric learning-based approach for video texture synthesis that overcomes the aforementioned limitations. As in [132], we synthesize textures by resampling frames from the input video. However, as opposed to using pixel similarity, we *learn* feature representations and a distance metric to compare frames by training a deep model on *a single input video*. The network is trained using contrastive learning to fit an example-specific bi-gram model (*i.e.* a Markov chain). This allows us to learn features that are spatially and temporally best suited to the input video.

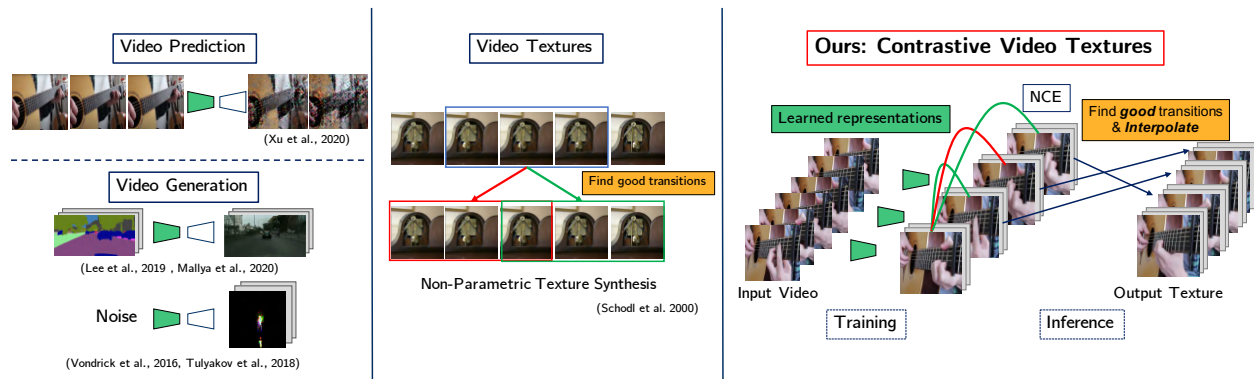


Figure 2.2: **Video Texture Synthesis.** Prior video prediction [170] and generation [154, 158] methods fail to generate long and diverse video textures at a high resolution. Vid2Vid [82, 100] methods require semantic maps as input and aren’t suitable for video texture synthesis. Classic video textures [132] (middle) can generate infinite sequences by resampling frames, but uses fixed representations which are not robust to varying domains. Our method (right) learns a representation and non-parametric method for infinite video texture synthesis based on resampling frames from an input video.

To synthesize the video texture, we use the video-specific model to compute probabilities of transitioning between frames of the same video. We represent the video as a graph where the individual frames are nodes and the edges represent transition probabilities predicted by our video-specific model. We generate output videos (or textures) by randomly traversing edges with high transition probabilities. Our proposed method is able to synthesize realistic, smooth, and diverse output textures on a variety of dance and music videos as shown in the Supp. and at this [website](#) for easy viewing. Fig. 2.2 illustrates the distinction between video generation/prediction, video textures, and our contrastive model.

Learning the feature representations allows us to easily extend our model to an audio-conditioned video synthesis task as seen in Fig. 2.1. Given a source video with associated audio and a new *conditioning* audio not in the source, we synthesize a new video that matches the conditioning audio. A demonstration of this task where the guitarist is “strumming to the beats” of a new song is included in the Supp. We modify the inference algorithm to include an additional constraint that the predicted frame’s audio should match the conditioning audio. We trade off between temporal coherence (frames predicted by the contrastive video texture model) and audio similarity (frames predicted by the audio matching algorithm) to generate videos that are temporally smooth and also align well with the conditioning audio.

We assess the quality of the synthesized textures by conducting human perceptual evaluations comparing our method to a number of baselines. In the case of unconditional video texture synthesis, we compare to the classic video texture algorithm [132] and variations to this which we describe in Sec. 4.5. For the audio-conditioning setting, we compare to four different baselines: classic video textures with audio-conditioning, visual rhythm and beat [23], Audio Nearest-Neighbours,

and a random baseline. Our studies confirm that our method is perceptually better than all of these previous methods.

2.2 Related Work

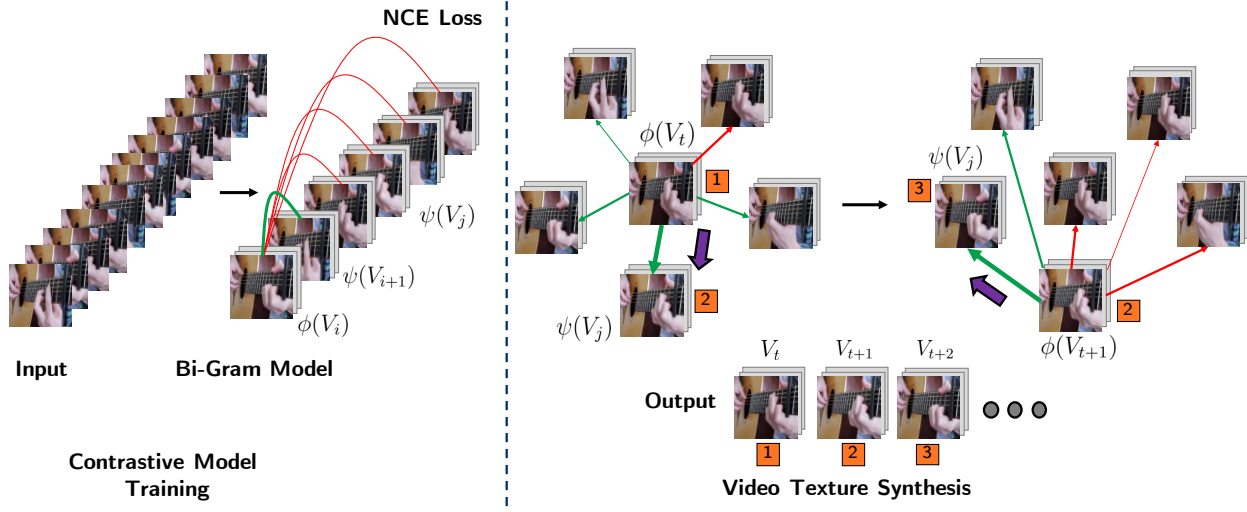


Figure 2.3: **Contrastive Video Textures.** We extract overlapping segments from the video and fit a bi-gram model trained using NCE loss (Eq. 2.2) which learns representations for query/target pairs such that given a query segment V_i , $\phi(V_i)$ is similar to positive segment $\psi(V_{i+1})$ and dissimilar to negative segments $\psi(V_j)$ where $j \in [1, \dots, N]$ and $j \neq i, i + 1$. **Video Texture Synthesis.** During inference, we start with a random segment V_t shown by **1**, compute $\phi(V_t)$ and $\psi(V_j) \forall j \in [1, \dots, N]$ and calculate the edge weights as similarity between $\phi(V_t)$ and $\psi(V_j)$. We denote higher weight edges in green and lower weighted edges in red and the thickness correlates with the probability. We randomly traverse (purple arrow) along one of the higher weighted edges to reach **2**. **1** and **2** are appended to the output and the process is repeated (orange arrow) with **2** as the query.

Texture Synthesis. All texture synthesis methods aim to produce textures which are sufficiently different from the source yet appear to be produced by the same underlying stochastic process. Texture synthesis methods can be broadly classified into two categories: non-parametric and parametric. Non-parametric methods focus on modeling the conditional distribution of the input images and sample information directly from the input. The sampling could be done pixel-wise [30, 166] or patch-wise [29, 80] for image texture synthesis. Wei *et al.* [167] provides an extensive review of example-based texture synthesis methods. Parametric approaches, on the other hand, focus on explicitly modeling the underlying texture synthesis process. Heeger *et al.* [54] and Portilla *et al.* [118] were the first to propose parametric image texture synthesis by matching statistics of image features between source and target images. This later inspired Gatys *et al.* [39], which used

features learned using a convolutional neural network for image texture synthesis. Inspired by these works, [132] proposed a non-parametric approach for synthesizing a video texture with by finding novel, plausible transitions in an input video. Following work [131, 130, 31] explored interesting extensions of the same. All these video texture synthesis works use Euclidean pixel distance as a similarity measure. This causes the texture synthesis to fail on more complex scenes. On the other hand, our learned contrastive feature representations and similarity metric generalizes well to dance/music domains and also allows for conditioning on heterogeneous data such as audio.

Video Generation and Video Prediction. The success of (GANs) [43] and Variational Autoencoders (VAEs) [73] in image generation [68, 116, 194] inspired several video generation methods, both unconditional [21, 58, 128, 154, 158] and conditional [20, 38, 100, 102, 160, 162, 181, 193]. While conditional video synthesis of future frame prediction given past frames [25, 65, 144, 170, 174] works well, these methods are far from generating realistic infinitely-long and diverse video. They oftentimes produce outputs which are low-resolution, especially in the unconditional case. This is because videos are higher dimensional and modeling spatio-temporal changes and transition dynamics is more complex. As such, these methods are expected to fail when applied to our task of video texture synthesis which involves rendering a video as an infinitely varying stream of images. This requires capturing the temporal dynamics of the video which current video generation methods fail to do. Similar to recent works which condition the video generation on an input signal such as text [90], or speech [32, 71, 110], or a single image [137], we condition video texture synthesis on an audio signal. Our work is inspired by test-time training methods such as SinGAN [137], Deep Image Prior [155], and Patch VAE-GAN [46] in that we train an example-specific model on a single input, though on a video instead of an image and without an adversarial loss. Our method only takes a few hours to train on a single video and doesn't require hours of training on a large dataset.

Contrastive Learning. Recent contrastive learning approaches [19, 20, 18, 52, 55] have achieved success in classic vision tasks proving the usefulness of the learned representations. Mishra *et al.* [106] train a network to determine the temporal ordering of frames in a video and Wei *et al.* [164]'s self-supervised model learns to tell if a video is playing forwards/backwards. Here, we use contrastive learning to fit a video-specific bi-gram model. Our network maximizes similarity between learned representations for the current and next frame. Unlike [111], our goal is not to generate frames from latent representations, but rather to use the learned distance metric to resample from the input video.

2.3 Contrastive Video Textures

An overview of our method is provided in Fig. 4.2. We propose a non-parametric learning-based approach for video texture synthesis. At a high-level, we fit an example-specific bi-gram model (*i.e.* a Markov chain) and use it to re-sample input frames, producing a diverse and temporally coherent video. In the following, we first define the bi-gram model, and then describe how to train and sample from it.

Given an input video, we extract N overlapping segments denoted by V_i where $i \in [1, \dots, N]$, with a sliding window of length W and stride s . Consider these segments to be the states of a Markov chain, where the probability of transition is computed by a deep similarity function parameterized by encoders ϕ and ψ :

$$P(V_{i+1}|V_i) \propto \exp(\text{sim}(\phi(V_i), \psi(V_{i+1}))/\tau) \quad (2.1)$$

We use two separate encoder heads ϕ and ψ for the query and target, respectively, to break the symmetry between the two embeddings. This ensures $\text{sim}(V_i, V_{i+1}) \neq \text{sim}(V_{i+1}, V_i)$, which allows the model to learn the arrow of time. Fitting the transition probabilities amounts to fitting the parameters of ϕ and ψ , which here will take form of a 3D convolutional network. The model is trained using temperature-scaled and normalized NCE Loss [105]:

$$\begin{aligned} \mathcal{L}(V, \phi) &= \sum_{i=1}^N -\log P(V_{i+1}|V_i) \\ &= \sum_{i=1}^N -\log \frac{\exp(S(V_i, V_{i+1})/\tau)}{\sum_{j=1}^N \mathbb{1}_{[j \notin \{i, i+1\}]} \exp(S(V_i, V_j)/\tau)} \\ &\text{where, } S(V_i, V_j) = \text{sim}(\phi(V_i), \psi(V_j)) \end{aligned} \quad (2.2)$$

where τ denotes a temperature term that modulates the sharpness of the softmax distribution. As the complexity increases with number of negatives in the denominator, for efficiency, we use negative sampling [105] to approximate the denominator in Eq 2.2. Fitting the encoder in this manner amounts to learning a video representation by contrastive learning, where the positive is the segment that follows, and negatives are sampled from the set of all other segments. The encoder thus learns features useful for predicting the dynamics of phenomena specific to the input video.

Video Texture Synthesis. To synthesize the texture, we represent the video as a graph, with nodes as segments and edges indicating the transition probabilities computed by our Contrastive model as shown in Fig. 4.2. We randomly select a query segment V_t among the segments of the video and set the output sequence to all the W frames in V_t . Next, our model computes $\phi(V_t)$ and $\psi(V_j)$ for all target segments in the video and updates the edges of the graph with the transition probabilities, given by $\text{sim}(\phi(V_t), \psi(V_j))$.

Given that we fit the model on a single video, it is important that we ensure there is enough entropy in the transition distribution in order to ensure diversity in samples synthesized during inference. Always selecting the target segment with the highest transition probability would regurgitate the original sequence, as the model was trained to predict V_{j+1} as the positive segment given V_j as the query. Thus, given the current segment V_j , while we could transition to the very next segment V_{j+1} , we want to encourage the model to transition to other segments similar to V_{j+1} . While we assume that our input video sequence exhibits sufficient hierarchical, periodic structure to ensure repetition and multi-modality, we can also directly adjust the conditional entropy of the model through the softmax temperature term τ . A lower temperature would flatten the transition probabilities (*i.e.* increase the entropy) and reduce the difference in probabilities of the positive segment and segments similar to it. To avoid abrupt and noisy transitions, we set all transition

probabilities below a certain threshold to zero. The threshold is set to be $t\%$ of the maximum transition probability connecting V_j to any other node V_t . We compute t heuristically and include details in Supp.

Next, we randomly select a positive segment to transition to from the edges with non-zero probabilities. This introduces variance in the generated textures and also ensures that the transitions are smooth and coherent. We then append the last s number of frames in the positive segment to the output. This predicted positive segment V_{t+1} is again fed into the network as the query and this is repeated to generate the whole output in an autoregressive fashion.

Video Encoding. We use the SlowFast [35] action recognition model pretrained on Kinetics-400 [69] for encoding the video segments. We include more details in Supp.

Interpolation. For smoother transitions, we also conditionally interpolate between frames of the synthesized texture when there are transitions to different parts of the video. We use a pre-trained interpolation network of Jiang *et al.* [63]. In Sec. 2.5, we include results both with and without interpolation to show that interpolation helps with smoothing.

2.4 Audio-Conditioned Video Textures

We show that it is easy to extend our Contrastive Video Textures algorithm to synthesize videos that match a conditioning audio signal. Given an input video with corresponding audio A^s and an external conditioning audio A^c , we synthesize a new video that is synchronized with the conditioning audio. We extract N overlapping segments from the input and conditioning audio, as before. We compute the similarity of the input audio segments A^s to the conditioning audio segment A^c by projecting them into a common embedding space. We construct a transition probability matrix T_a in the audio space as,

$$T_a(i, j) = \text{sim}(\varphi(A_i^c), \varphi(A_j^s))$$

Note that, unlike video segments in Eq. 2.1, the audio segments come from two separate audio signals. Hence, there’s no need to have two separate subnetworks as there’s no symmetry and we use the same audio encoder φ for both. We compute the transition probabilities T_v for the target video segments given the previous predicted segment using the Contrastive video textures model (Eq. 2.2). The joint transition probabilities for a segment are formulated as a trade-off between the audio-conditioning signal and the temporal coherence constraint as,

$$T = \alpha T_v + (1 - \alpha) T_a \tag{2.3}$$

Audio Encoding. We embed the audio segments using the VGGish model [56] pretrained on AudioSet [40].

We describe the implementation details of our method and hyperparameter choices in Supp.

Method	Preference %
Classic	3.33 ± 2.42 %
Classic Deep	6.66 ± 3.37 %
Classic+	10.95 ± 4.22 %
Classic++	9.52 ± 3.97 %
1-4 Any Classic	30.48 ± 6.22 %
Contrastive	69.52 ± 6.22 %

Table 2.1: **Perceptual Studies for Unconditional Video Textures.** We show MTurk evaluators textures synthesized by all 5 methods and ask them to pick the most realistic one. We also report the chance evaluators chose *any* of the variation of the classic model.

Method	Real vs. Fake
Classic++	$11.4 \pm 4.30\%$
Classic+	15.7 ± 4.92 %
Contrastive	$25.7 \pm 4.30\%$

Table 2.2: **Unconditional: Real vs. Fake study.** We show evaluators a pair of videos (generated and real video) without labels, ask them to pick the real one. Our method fools evaluators more times than Classic.

2.5 Experiments

We curate a dataset of 70 videos from different domains such as dance and musical instruments including piano, guitar, suitar, tabla, flute, ukelele, and harmonium. A subset of these videos were randomly sampled from the PianoYT dataset [74] and the rest were downloaded from YouTube. We used 40 (of 70) videos to tune our hyperparameters and tested on the remaining 30 with no additional tuning. Our dataset consists of both short videos which are 2-3 minutes long and long videos ranging from 30-60 mins¹. We conduct perceptual evaluations on Amazon MTurk to qualitatively compare the results from our method to different baselines for both the unconditional and conditional settings. We also include results of ablating the interpolation module. Additionally, we introduce and report results on a new metric, diversity score, which measures the diversity of the textures.

Unconditional Video Texture Synthesis

To show the effectiveness of our method, we compare our results to the Classic video textures algorithm [132] and its three variations. The algorithm and its variants are described in Sec. 1 of Supplementary. Classic+, like Contrastive, appends multiple frames to the output sequence instead

¹We will release the videos, trained models, and a reference implementation of our method.

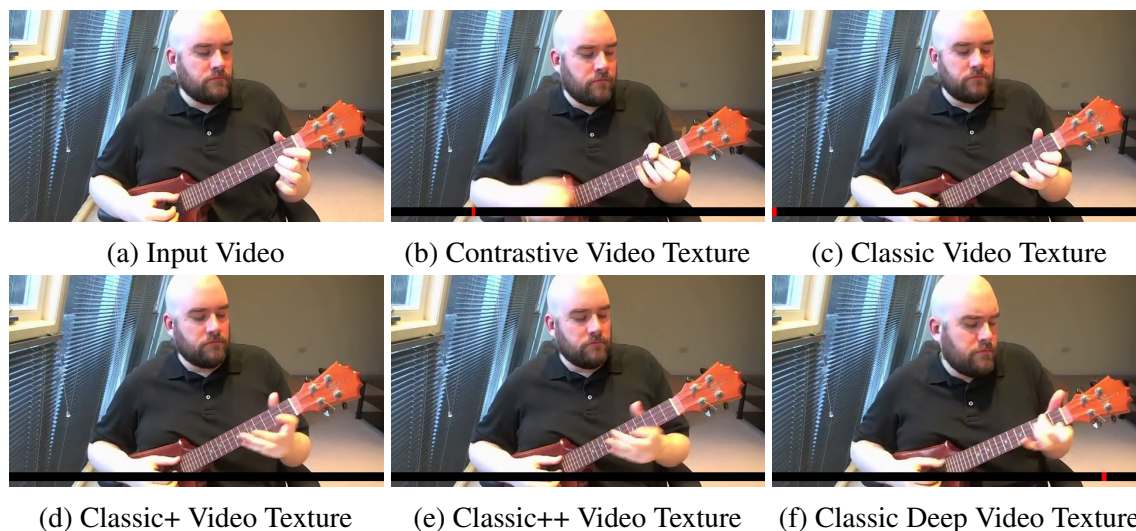


Figure 2.4: **Unconditional Contrastive Video Textures.** Click on each figure to play the video. The figure shows the input video and textures synthesized using our Contrastive method and the baselines Classic, Classic+, Classic++, and Classic Deep. The red bar at the bottom of each video indicates the part of the input video being played. Classic, Classic+, and Classic++ textures loop over the same frame at the start of the video as shown by the red bar, are choppy, and not diverse. Classic Deep texture has jarring transitions. Our Contrastive method finds smooth and seamless transitions in the video to produce a texture that’s diverse yet dynamically consistent.

Method	Real vs Fake
Random Clip	$15.33 \pm 5.76\%$
Audio NN	$20.4 \pm 6.63\%$
Contrastive	$26.74 \pm 6.14\%$

Table 2.3: **Conditional: Real vs. Fake study.** We show evaluators a pair of videos (generated and real video) without labels and ask them to pick the real one. Our method fooled evaluators more often than the baselines.

of a single frame, Classic++ adds a stride while filtering the distance matrix and Classic Deep uses ImageNet pretrained ResNet features instead of raw pixel values. For fairness, we added the interpolation module described in Sec. 2.3 to all the baselines.

Table 2.1 reports the results from a perceptual study on Amazon MTurk where evaluators were shown textures generated by all five methods and asked to choose the one they found most realistic. Our Contrastive model surpasses all baselines by a large margin and was chosen 69.52% of the time. Since the classic models are similar, we also report all variations of classic combined. They are chosen 30.48% of the time.

We include qualitative video results for Contrastive, Classic, Classic+, Classic++, and Classic Deep in Fig. 2.4. The red bar at the bottom of each video indicates the part of the input video being played. Video in Fig. 2.4b produced by our Contrastive method is the most realistic and consistent with seamless transitions. The red bar transitioning to new positions indicates that our textures are dynamic and diverse (vary over time). Classic in Fig. 2.4c loops over the same set of frames in and around the target, thus appearing stuck. Classic+ and Classic++ shown in Fig. 2.4d and Fig. 2.4e have slightly improved quality compared to it but lack diversity and produce jarring transitions. Classic Deep texture in Fig. 2.4f is choppy due to multiple poor transitions chosen by the model.

Additionally, we conduct real vs. fake studies in Table 2.2 where the evaluators are shown the ground truth video and synthesized texture and asked to pick the one they think is real. Our method is able to fool evaluators 25.7% of the time whereas the best baseline (Classic+) is able to fool the evaluators only 15.7% of the time.

Both the qualitative and quantitative comparisons clearly highlight the issues with the Classic model and emphasize the need to learn the feature representations and the distance metric as we do in our Contrastive method. Fig. 2.5 shows more qualitative results of Unconditional Contrastive Video Textures on videos of guitar, dance, and Indian musical instruments. Our method works well on all the domains and produces dynamic yet consistent video textures. The change in position of the red bar indicates that our method seamlessly transitions across different parts of the input video. As seen in the dance videos, learned representations result in transitions that are consistent with the arm movements of the dancer.

Diversity. For a fair comparison, we set the temperature for both Contrastive and Classic+ methods such that the resulting videos have approximately the same number of transitions. We synthesize Classic+ and Contrastive video textures with 9 ± 2 transitions each for 20 videos. Evaluators were shown textures from both methods and asked to pick the one they found more realistic. Contrastive videos were preferred 76.6% of the time, comparable to the result in Tab. 2.1, indicating that our method finds better transitions. Additionally, we measure diversity score (DS) as the number of new transitions in every 30 seconds of the synthesized video, averaged over all videos. A transition is considered *new* if it hasn't occurred in the 30 second time-frame. Our method achieves a DS of 7.78, indicating that our textures are diverse and contain, on an average, 7 (of 9) new transitions every 30 seconds. Classic+ achieves a DS of 2.3 indicating that the video texture loops over the same part of the input video.

Interpolation. We verify the effectiveness of the interpolation module through a perceptual study. Evaluators were shown two videos (with and without interpolation), and asked to pick the one they found more realistic. They picked the video with interpolation 89% of the time, thus confirming that interpolation leads to an improvement in perceptual quality. We show qualitative comparisons in Supp. (and [here](#)).

Audio-Conditioned Contrastive Video Textures

For audio-conditioned video synthesis, we randomly paired the 70 videos with songs from the same domain (*e.g.* an input piano video is paired with a conditioning audio of a piano). Using this strategy, we created 70 input video-conditioning audio pairs. As described in Sec. 2.4, we extend

our Contrastive method to synthesize textures given a conditioning audio signal. We compare audio-conditioned video textures synthesized by our method to four baselines and report results from a perceptual evaluation.

- *Random Clip*. In this baseline, we choose a random portion of the input video to match the conditioning audio.
- *Classic+Audio*. We add audio-conditioning to the classic video textures algorithm. For this, we divide the conditioning audio into segments and find nearest neighbours in the input audio. Then we combine these distances with the distance matrix calculated by video textures using Eq. 2.3.
- *Visual Rhythm and Beat (VRB)*. We use the approach of Davis *et al.* [23] to synchronize the input video with the audio beats. This method works by changing the pace of the video to better align the visual and audio beats.
- *Audio Nearest Neighbours*. We include comparisons to the nearest neighbor baseline that works by computing the similarity between the conditioning audio signal and segments of the input audio of the same length and then choosing the video clip of the closest match.

Fig. 2.6 shows results from our perceptual studies comparing the audio-conditioned video textures synthesized by our Contrastive model to all of the baselines. The evaluators were shown two videos with the same conditioning audio, one synthesized by our method and the other by the corresponding baseline. They were asked to pick the video that was more in sync with the audio. Our method outperforms all baselines by a large margin. As shown in Tab. 2.3, we conducted a real vs. fake study comparing the ground truth videos with the synthesized videos from Contrastive and the two best baselines (Random Clip and Audio NN). While Random Clip and Audio NN beat the ground truth only 15.33% and 20.4% respectively, our method was able to fool evaluators 26.74% of the time.

Fig. 2.7 shows qualitative results comparing our method to the five baselines described above and also include the videos in Supp. and [here](#). In Fig. 2.7 (A), the conditioning audio signal has a gap/break in the audio where no music is being played. We see the output produced by our Contrastive model is semantically meaningful and aligns best with the audio. Random Clip chooses a random segment which has strumming and thus fails to align with the audio. Similarly Classic+Audio chooses frames that don't correlate with the audio. VRB doesn't capture semantics as it only speeds up or slows down the video to better match the audio beats. Similarly, in Fig. 2.7(B) we see that our Contrastive method is able to pick up on repeated chords in the conditioning audio signal while no other method is able to do that. Through more examples listed in Supp. and at [this website](#), we show that the videos synthesized by Contrastive model are more in sync with the conditioning audio. For example, it identifies gaps in the audio, repeated chords, and change of pace. We observed experimentally that our method doesn't work well for videos where the scene constantly changes (such as waves) and where subtle asynchronies between audio and video are easy to spot (such as people speaking) as these applications are beyond the scope of video textures. We hope that they can be addressed with hybrid approaches that combine the benefits of video textures with GANs.

Comparison to Video Generation Methods

GAN based video generation methods cannot capture the temporal dynamics of the video and thus fail to synthesize long and diverse textures. We compare our unconditional Contrastive method to MoCoGAN [154], an unconditional video generation method and include results in Supp. (and [here](#)). A 3-second video of a candle flame is given as input and our method is able to produce a 30-second high resolution, temporally consistent, and diverse output of a candle flickering. On the other hand, MoCoGAN’s video contains artifacts and the flame lasts for only 3 seconds. Similarly, with the guitar video, our method produces a realistic video with seamless transitions whereas MoCoGAN’s output is blurry.

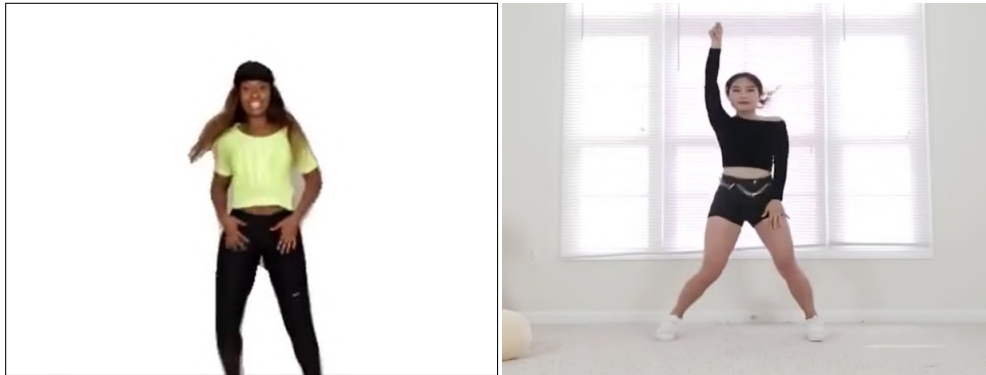
2.6 Conclusion

We presented Contrastive Video Textures, a learning-based approach for video textures applied to audio-conditioned video synthesis. Our method fits an input-specific bi-gram model to capture the dynamics of a video, and uses it to generate diverse and temporally coherent textures. We also introduced audio-conditioned video texture synthesis as a useful application of video textures. We show that our model outperforms a number of baselines on perceptual studies.

Acknowledgements. We thank Arun Mallya, Allan Jabri, Anna Rohrbach, Amir Bar, Suzie Petryk, and Parsa Mahmoudieh for very helpful discussions and feedback. This work was supported in part by DoD including DARPA’s XAI, LwLL, and SemaFor programs, as well as BAIR’s industrial alliance programs.



(a) Guitar



(b) Dance



(c) Indian Musical Instruments (Tabla and Sitar)



(d) Harp

Figure 2.5: Qualitative Results of **Unconditional Contrastive Video Textures**. Click on the image to play the video.

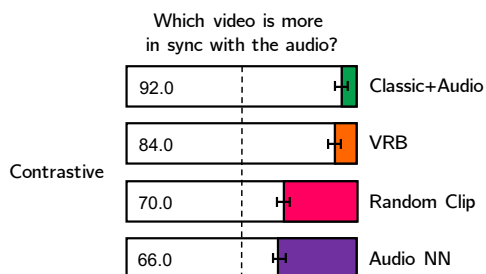


Figure 2.6: **Perceptual Studies for Audio-Conditioned Contrastive Video Textures.** We compare results from our Contrastive method against each of the baselines, individually. Evaluators were shown two videos, one synthesized by our method and the other by the corresponding baseline and asked to pick the one where the audio and video were more in sync.

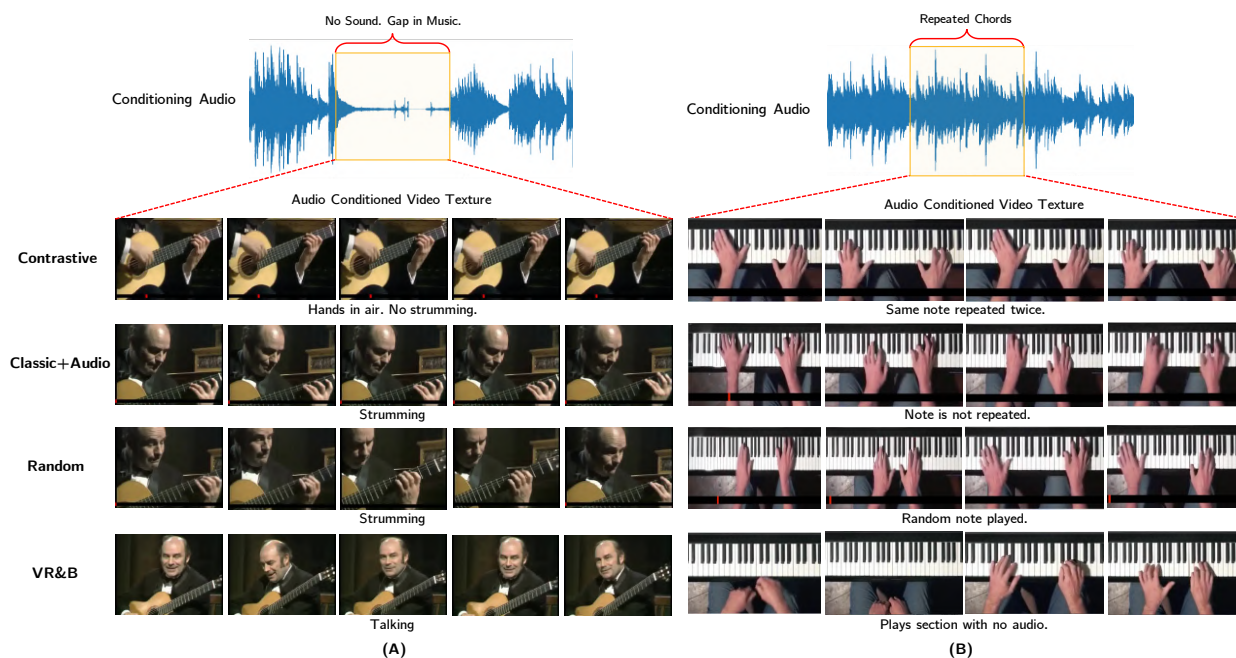


Figure 2.7: Qualitative comparison of audio-conditioned video textures synthesized by Classic+Audio, Random Clip, Visual Rhythm and Beat (VRB) and our Contrastive model. **(A)** The conditioning audio waveform shows a gap in the audio where no music is being played. Our model is able to pick up on that and the corresponding video that is synthesized has hands in the air and no strumming. However, both Random Clip and Classic+Audio show strumming, and VRB shows the person talking. **(B)** The conditioning audio waveform has the same chord repeated twice. The video synthesized by our model reflects this, and we observe the same frames (1 and 2) repeated again. Classic+Audio and Random Clip don't repeat the note and VRB result contains a region without audio where the person isn't playing anything.

Chapter 3

CLIP-It! Language-Guided Video Summarization

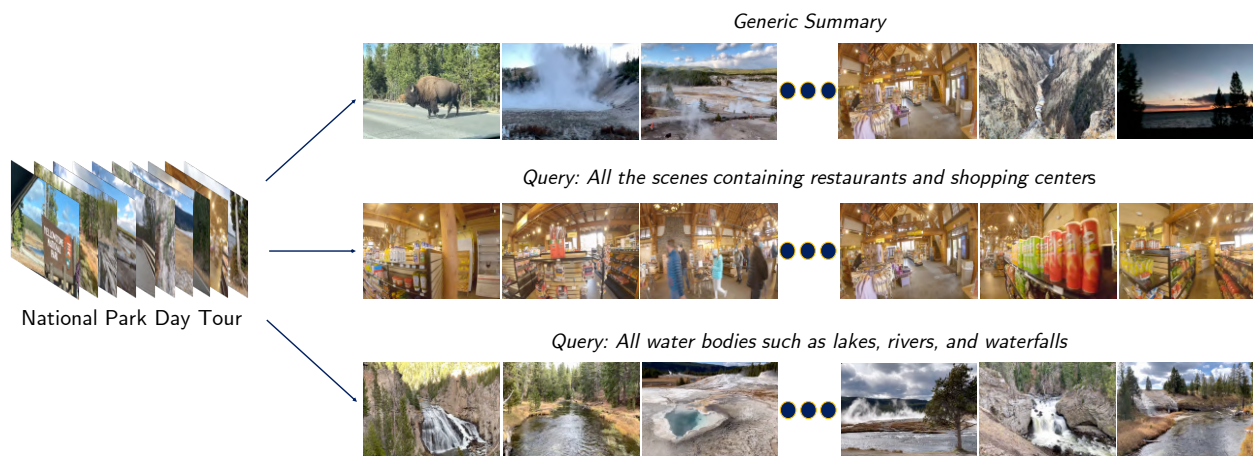


Figure 3.1: We introduce CLIP-It, a *language-guided* multimodal transformer for generic and query-focused video summarization. The figure shows the results from our method. Given a day-long video of a national park tour, the generic summary (top) is a video with relevant and diverse keyframes. When using the query “All the scenes containing restaurants and shopping centers”, the generated query-focused summary includes all the matching scenes. Similarly, the query “All water bodies such as lakes, rivers, and waterfalls”, yields a short summary containing all the water bodies present in the video.

3.1 Introduction

An effective video summary captures the essence of the video and provides a quick overview as an alternative to viewing the whole video; it should be succinct yet representative of the entire video.

This chapter is based on joint work with Anna Rohrbach and Trevor Darrell, and is presented much as it appeared in the NeurIPS 2021 proceedings.

Summarizing videos has many use cases - for example, viewers on YouTube may want to watch a short summary of the video to assess its relevance. While a generic summary is useful for capturing the important scenes in a video, it is even more practical if the summary can be customized by the user. As seen in Fig. 4.1, users should be able to indicate the concepts of the video they would like to see in the summary using natural language queries.

Generic video summarization datasets such as TVSum [143] and SumMe [48] provide ground-truth annotations in the form of frame or shot-level importance scores specified by multiple annotators. Several learning-based methods reduce the task to a frame-wise score prediction problem. Sequence labeling methods [42, 183, 184, 182, 89] model variable-range dependencies between frames but fail to capture relationships across all frames simultaneously. While attention [33] and graph based [115] methods address this partly, they disregard ordering of the input frames which is useful when predicting scores. Moreover, these methods use only visual cues to produce the summaries and cannot be customized with a natural language input. Another line of work, query-focused video summarization [138], allows the users to customize the summary by specifying a query containing two concepts (eg., food and drinks). However, in their work the query can only be chosen from a fixed set of predefined concepts which limits the flexibility for the user.

It is important to note that efforts in generic and query-focused video summarization have so far been disjoint, with no single method for both. Our key innovation is to unify these tasks in one *language-guided* framework. We introduce *CLIP-It*, a multimodal summarization model which takes two inputs, a video and a natural language text, and synthesizes a summary video conditioned on the text. In case of generic video summarization, the natural language text is a video description obtained using an off-the-shelf dense video captioning method. Alternatively, in the case of query-focused video summarization, the language input is a user-defined query. Unlike existing generic methods which only use visual cues, we show that adding language as an input leads to the “discovery” of relevant concepts and actions resulting in better summaries. Our method uses a Transformer with positional encoding, which can attend to all frames at once (unlike LSTM based methods) and also keep track of the ordering of frames (unlike graph based methods). In contrast to existing query-focused methods [138] which only allow keyword based queries, we aim to enable open-ended natural language queries for users to customize video summaries. For example, as seen in Fig. 4.1, using our method users can specify long and descriptive queries such as, “All water bodies such as lakes, rivers, and waterfalls” which is not possible with previous methods.

Given an input video, CLIP-It generates a video summary guided by either a user-defined natural language query or a system generated description. It uses a *Language-Guided Attention* head to compute a joint representation of the image and language embeddings, and a *Frame-Scoring Transformer* to assign scores to individual frames in the video using the fused representations. Following [184, 183], the summary video is constructed from high scoring frames by converting frame-level scores to shot-level scores and using knapsack algorithm to fit the maximum number of high scoring shots in a timed window. Fig. 4.1 shows an output from our method. Given an hour long video of a national park tour, we generate a 2 minute generic summary comprising of all the important scenes in the video. Given the two language queries, our method picks the matching keyframes in both cases. We can train CLIP-It without ground-truth supervision by leveraging the reconstruction and diversity losses [53, 127]. For generic video summarization, we evaluate our

approach on the standard benchmarks, TVSum and SumMe. We achieve performance improvement on F1 score of nearly 3% in the supervised setting and 4% in the unsupervised setting on both datasets. We show large gains (5%) in the Transfer setting, where CLIP-It is trained and evaluated on disjoint sets of data. For the query-focused scenario we evaluate on the QFVS dataset [139], where we also achieve state-of-the-art results.

To summarize our contributions, we introduce CLIP-It, a *language-guided* model that unifies generic and query-focused video summarization. Our approach uses language conditioning in the form of off-the-shelf video descriptions (for generic summarization) or user-defined natural language queries (for query-focused summarization). We show that the Transformer design enables effective contextualization across frames, benefiting our tasks. We also demonstrate the impact of language guidance on generic summarization. Finally, we establish the new state-of-the-art on both generic and query-focused datasets in supervised and unsupervised settings.

3.2 Related Work

Generic Video Summarization. A video summary is a short synopsis created by stitching together important clips from the original video. Early works [98, 140, 141] referred to it as Video Skimming or Dynamic Video Summarization and used hand-designed features to generate summaries. Likewise, non-parametric unsupervised methods [67, 83, 109, 94, 97, 119] used various heuristics to represent the importance of frames. Introduction of benchmark datasets such as TVSum [143] and SumMe [48] provided relevance scores for frames in videos annotated by users, resulting in multiple human generated summaries. This enabled automatic evaluation of video summarization techniques and has led to the development of many supervised learning based methods [47, 50, 99, 115, 127, 126, 176, 183, 184, 182, 186]. These approaches capture high-level semantic information and outperform the heuristic unsupervised methods. Fully convolutional sequence networks [127] treat video summarization as a binary label prediction problem. Determinantal point processes [81] and LSTM [57] based approaches [42, 89, 183, 184, 182] model variable-range dependencies between frames. However, these are sequential and fail to capture relationships across all frames simultaneously. Attention [33] and graph based methods [115] address this issue by modeling relationships across all frames, but they disregard the ordering of frames in a video, which is also useful for summarization. Our method uses a Transformer [156] with positional encoding, which allows for joint attention across all frames while maintaining an ordering of the input sequence of frames.

Some of the above works have supervised and unsupervised variants with modifications to the objective function. Specifically, for the unsupervised variant, they use reconstruction and diversity losses which do not require ground truth. We follow prior work in terms of using the same loss functions. Other notable unsupervised approaches include an adversarial LSTM based method [99], a generative adversarial network to learn from unpaired data [126], and a cycle consistent learning objective [176].

Video-Text Summarization. Plummer *et al.* [117] use image-language embeddings for generating video summaries but evaluate their approach in the text domain, and not on the generic

video summarization benchmarks. Furthermore, they require text to be provided as input and don't have a mechanism to generate captions if absent. On the other hand, our method works well with off-the-shelf captions and we evaluate on both generic and query-focused benchmark datasets. Chen *et al.* [16] jointly train a text and video summarization network but rely on ground-truth text summaries.

Query-Focused Video Summarization. Oftentimes users browsing videos on YouTube are looking for something specific so a generic summary might not suffice. In this case, there should be an option to customize the generated summary using a natural language query. Sharghi *et al.* [138] introduce the Query-Focused Video Summarization (QFVS) dataset for UT Egocentric [83] videos containing user-defined video summaries for a set of pre-defined concepts. Sharghi *et al.* [139] propose a memory network to attend over different video frames and shots with the user query as input. However, this recurrent attention mechanism precludes parallelization and limits modeling long-range dependencies, which is overcome by our Transformer architecture. Moreover, their method only works with the pre-defined set of keyword based queries in QFVS dataset. Since we use CLIP [123] to encode the language input and train our method on dense video descriptions, this allows users to define freeform queries at test time (as seen in Fig. 4.1). Other works [66, 165] similarly condition the summary generation on keyword based queries but haven't released their data.

Transformers. Transformers [156] were introduced for neural machine translation and have since been applied to video-language tasks such as video-retrieval [37] and video captioning [85, 192]. In this work we adapt transformers for video summarization. We modify self-attention [156] to a Language-Guided Attention block that accepts inputs from two modalities. Additionally, our method also relies on CLIP [123] for extracting image and text features and a Bi-Modal Transformer [62] for dense video caption generation, both of which also have transformer backbone.

3.3 CLIP-It: Language-Guided Video Summarization

CLIP-It is a unified *language-guided* framework for both generic and query-focused video summarization. It takes an input video and a user-defined query or a system generated dense video caption and constructs a summary by selecting key frames from the video. First, we explain the intuition behind our approach. In the case of query-focused summarization, clearly, it is necessary to model the user query as input in order to produce an appropriate summary. In the case of generic video summarization no user query is available; nonetheless, we show here that we can leverage the semantic information contained in associated natural language descriptions to guide the summarization process. Assuming we had a generic description that accompanies a video (e.g., *A person is walking a dog. The person throws a ball. The dog runs after it.*), we could leverage its semantic embedding to *match it* to the most relevant portions of the video. Such a description could be obtained automatically by generating dense video captions [62]. We first present an overview of our approach, CLIP-It, followed by a detailed description of the individual components.

Overview. Our approach, CLIP-It, is outlined in Fig 4.2. Given a video, we extract N frames denoted by $F_i, i \in [1, \dots N]$. We formulate the video summarization task as a per-frame binary

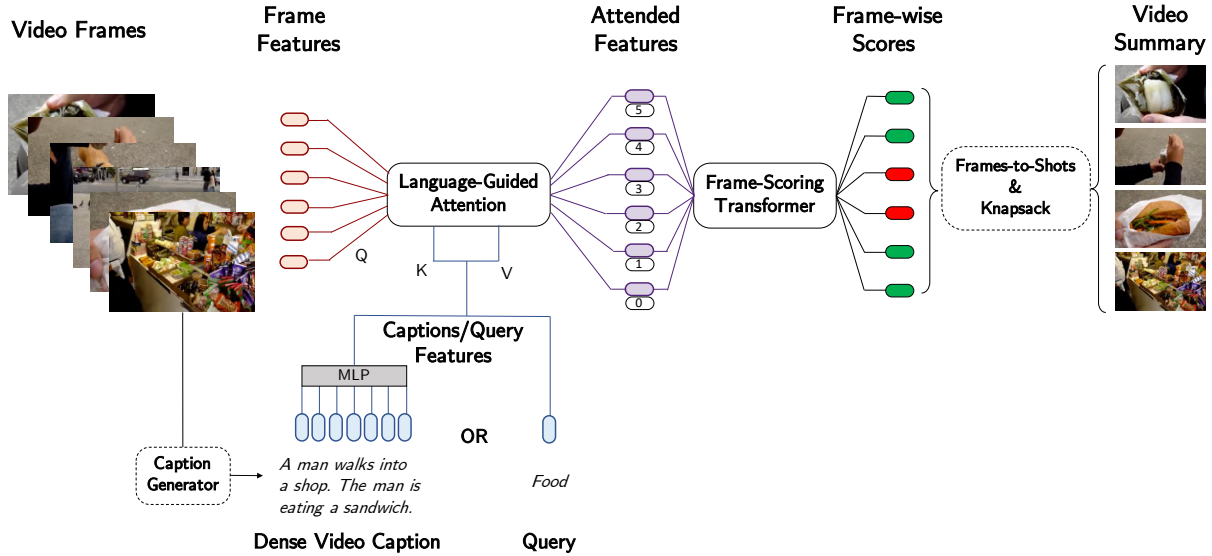


Figure 3.2: **Overview of CLIP-It.** Given an input video, CLIP-It generates a summary conditioned on either a user-defined natural language query or an automatically generated dense video caption. The Language-Guided Attention head fuses the image and language embeddings and the Frame-Scoring Transformer jointly attends to all frames to predict their relevance scores. During inference, the video summary is constructed by converting frame scores to shot scores and using Knapsack algorithm to select high scoring shots.

classification problem. We embed the frames using a pretrained network f_{img} . If a query is provided (in the form of a natural language string), we embed the query using a pretrained network f_{txt} . Alternatively, as seen in the figure, we use an off-the-shelf video captioning model to generate a dense video caption with M sentences denoted by C_j , $j \in [1, \dots, M]$ and embed each sentence using the pretrained network f_{txt} . Next, we compute language attended image embeddings I^* using learned Language-Guided Multi-head Attention $f_{img_txt}^*$. Finally, we train a Frame-Scoring Transformer which assigns scores to each frame in the video (green indicating a high score and red indicating a low score). To construct the video summary during inference, we convert frame-level scores to shot-level scores and finally, use 0/1 knapsack algorithm to pick the key shots [184]. In the following, we describe the Language-Guided Attention and the Frame-Selection Transformer modules, followed by discussing the visual and language encoders.

Language-Guided Attention. We use Language-Guided Multi-head Attention to efficiently fuse information across the video and language modalities and to infer long-term dependencies across both. Using a single attention head does not suffice as our goal is to allow all captions to attend to all frames in the video. We modify the Multi-Head Attention described in Vaswani *et al.* [156] to take in inputs from both modalities. We set Query Q , Key K , and Value V as follows,

$$\begin{aligned}
Q &= f_{img}(F_i), \text{ where } i \in [1, \dots, N], \\
K, V &= f_{txt}(C_j), \text{ where } j \in [1, \dots, M], \\
\text{Language - Guided Attn.}(Q, K, V) &= \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O, \\
\text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \\
\text{and Attention}(Q, K, V) &= \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V
\end{aligned}$$

W_i^Q , W_i^K , and W_i^V are learned parameter matrices and d_k is the dimensions of K . The output of the Language-Guided Multi-Head Attention are the attended image embeddings, denoted as F'_i .

Frame-Scoring Transformer. Finally, it is also important to ensure that we do not include redundant information, e.g., several key shots from the same event. To better model interactions across frames and contextualize them w.r.t. each other, we add a Frame-Scoring Transformer that takes image-text representations as input and outputs one score per frame. Based on the Transformer model [156], this module assigns relevance scores to the attended image embeddings F'_i . We feed F'_i to the bottom of both the encoder and decoder stacks. Similar to [156], we use positional encoding to insert information about the relative positions of the tokens in the sequence. We add positional encodings to the input embeddings at the bottom of the encoder and decoder stacks.

Image Encoding. We encode the image using a pre-trained network f_{img} . We experiment with the following networks: GoogleNet [149] (for a fair comparison to prior work), ResNet [50], and CLIP [123].

Text Encoding. We encode the user-defined query or the system generated dense caption using a pre-trained network f_{txt} . In this case we tried the CLIP (ViT and RN101) model. In case of generic video summarization, we generate dense video captions for the *whole video* in order to condition on language and incorporate added semantics into our video summarization pipeline. We use the Bi-Modal Transformer [62] dense video captioning model that generates a multi-sentence description given a video with audio. Since there are multiple sentences in the dense video caption, we first embed each sentence of the caption using the text encoder f_{txt} described above. They are then concatenated and fused using a multi-layer perceptron (MLP).

Learning

We employ 3 loss functions (classification, diversity, and reconstruction) to train our model. The supervised setting uses all 3 and the unsupervised setting uses only diversity and reconstruction losses.

Classification loss. We use a weighted binary cross entropy loss for classifying each frame:

$$\mathcal{L}_c = -\frac{1}{N} \sum_{i=1}^N w^* [x_i^* \log(x_i)] + (1 - w^*) [(1 - x_i^*) \log(1 - x_i)], \quad (3.1)$$

where x_i^* is the ground-truth label of the i -th frame and N is the total number of frames in the video. w^* is the weight assigned to the class x_i^* , which is set to $\frac{\#keyframes}{N}$ if x_i^* is a keyframe and $1 - \frac{\#keyframes}{N}$ if x_i^* is a background frame.

For the purpose of training without any supervision, we employ two additional losses that enforce diversity in the selected keyframes. We first select keyframes X based on the scores assigned by the Frame-Scoring Transformer. We pass the output features of the transformer model for the selected keyframes through a decoder network consisting of 1×1 convolution layers to obtain reconstructed feature vectors for the selected keyframes such that each keyframe feature vector is of the same dimension as its corresponding input frame-level feature vector.

Reconstruction Loss. The reconstruction loss \mathcal{L}_r is defined as the mean squared error between the reconstructed features and the original features corresponding to the selected keyframes, such that:

$$\mathcal{L}_r = \frac{1}{X} \sum_{i \in X} \|\mathbf{x}_i - \hat{\mathbf{y}}_i\|_2, \quad (3.2)$$

where $\hat{\mathbf{y}}$ denotes the reconstructed features.

Diversity Loss. We employ a repelling regularizer [186] to enforce diversity among selected keyframes. Similar to [127, 126], we compute the diversity loss, \mathcal{L}_d , as the pairwise cosine similarity between the selected keyframes:

$$\mathcal{L}_d = \frac{1}{X(X-1)} \sum_{i \in X} \sum_{j \in X, j \neq i} \frac{\hat{\mathbf{y}}_i \cdot \hat{\mathbf{y}}_j}{\|\hat{\mathbf{y}}_i\|_2 \cdot \|\hat{\mathbf{y}}_j\|_2}, \quad (3.3)$$

where $\hat{\mathbf{y}}_i$ and $\hat{\mathbf{y}}_j$ denote the reconstructed feature vectors of the i -th and j -th node.

The final loss function for supervised learning is then,

$$\mathcal{L}_{sup} = \alpha \cdot \mathcal{L}_c + \beta \cdot \mathcal{L}_d + \lambda \cdot \mathcal{L}_r, \quad (3.4)$$

where α , β , and λ control the trade-off between the three loss functions. We modify the loss function to extend CLIP-It to the unsupervised video summarization setting. We omit \mathcal{L}_c since the groundtruth summary cannot be used for supervision and represent the final loss function for unsupervised learning as:

$$\mathcal{L}_{unsup} = \beta \cdot \mathcal{L}_d + \lambda \cdot \mathcal{L}_r, \quad (3.5)$$

where β and λ are balancing parameters to control the trade-off between the two terms. We include implementation details of our method in the Supp.

3.4 Experiments

In this section, we describe the experimental setup and evaluation of our method on two tasks: generic video summarization and query-focused video summarization.

Table 3.1: **Supervised.** Comparing F1 Scores of our methods with supervised baselines on the SumMe [48] and TVSum [143] datasets using Standard, Augment, and Transfer data configurations.

Method	SumMe			TVSum		
	Standard	Augment	Transfer	Standard	Augment	Transfer
Zhang <i>et al.</i> (SumTransfer) [183]	40.9	41.3	38.5	-	-	-
Zhang <i>et al.</i> (LSTM) [184]	38.6	42.9	41.8	54.7	59.6	58.7
Mahasseni <i>et al.</i> (SUM-GAN _{sup}) [99]	41.7	43.6	-	56.3	61.2	-
Rochan <i>et al.</i> (SUM-FCN) [127]	47.5	51.1	44.1	56.8	59.2	58.2
Rochan <i>et al.</i> (SUM-DeepLab) [127]	48.8	50.2	45.0	58.4	59.1	57.4
Zhou <i>et al.</i> [190]	42.1	43.9	42.6	58.1	59.8	58.9
Zhang <i>et al.</i> [182]	-	44.9	-	-	63.9	-
Fajtl <i>et al.</i> [33]	49.7	51.1	-	61.4	62.4	-
Rochan <i>et al.</i> [126]	-	48.0	41.6	-	56.1	55.7
Chen <i>et al.</i> (V2TS) [16]	-	-	-	62.1	-	-
He <i>et al.</i> [53]	47.2	-	-	59.4	-	-
Park <i>et al.</i> (SumGraph) [115]	51.4	52.9	48.7	63.9	65.8	60.5
GoogleNet+bi-LSTM	38.5	42.4	40.7	53.9	59.6	58.6
ResNet+bi-LSTM	39.4	44.0	42.6	55.0	61.0	59.9
CLIP-Image+bi-LSTM	41.1	45.9	44.9	56.8	63.7	61.6
CLIP-Image+Video Caption+bi-LSTM	41.2	46.1	45.5	57.1	64.3	62.4
GoogleNet+Transformer	51.6	53.5	49.4	64.2	66.3	61.3
ResNet+Transformer	52.8	54.9	50.3	65.0	67.5	62.8
CLIP-Image+Transformer	53.5	55.3	51.0	65.5	68.1	63.4
CLIP-It: CLIP-Image+Video Caption+Transformer	54.2	56.4	51.9	66.3	69.0	65.5

Generic Video Summarization

Generic video summarization involves generating a single general-purpose summary to describe the input video. Note that while prior works only use visual cues from the video, our method also allows for video captions as an input feature. For a fair comparison, we include ablations of our method that do not use any language cues and only the visual features used in earlier works [184].

Datasets. We evaluate our approach on two standard video summarization datasets (TVSum [143] and SumMe [48]) and on the generic summaries for UT Egocentric videos [83] provided by the QFVS dataset [139]. TVSum [143] consists of 50 videos pertaining to 10 categories (how to videos, news, documentary, etc) with 5 videos from each category, typically 1-5 minutes in length. SumMe [48] consists of 25 videos capturing multiple events such as cooking and sports, and the lengths of the videos vary from 1 to 6 minutes. In addition to training on each dataset independently, we follow prior work and augment training data with 39 videos from the YouTube dataset [24] and 50 videos from the Open Video Project (OVP) dataset [112]. YouTube dataset consists of news, sports and cartoon videos. OVP dataset consists of multiple different genres including documentary videos.

Table 3.2: **Unsupervised.** Comparing F1 Scores of our methods with unsupervised baselines on the SumMe [48] and TVSum [143] datasets using Standard, Augment, and Transfer data configurations.

Method	SumMe			TVSum		
	Standard	Augment	Transfer	Standard	Augment	Transfer
Mahasseni <i>et al.</i> [99]	39.1	43.4	-	51.7	59.5	-
Yuan <i>et al.</i> [176]	41.9	-	-	57.6	-	-
Rochan <i>et al.</i> (SUM-FCN _{unsup}) [127]	41.5	-	39.5	52.7	-	-
Rochan <i>et al.</i> [126]	47.5	-	41.6	55.6	-	55.7
He <i>et al.</i> [53]	46.0	47.0	44.5	58.5	58.9	57.8
Park <i>et al.</i> (SumGraph) [115]	49.8	52.1	47.0	59.3	61.2	57.6
GoogleNet+bi-LSTM	33.1	38.0	36.5	47.7	54.9	52.3
ResNet+bi-LSTM	34.5	40.1	39.6	51.0	56.2	53.8
CLIP-Image+bi-LSTM	35.7	41.0	41.4	52.8	58.7	56.0
CLIP-Image+Video Caption+bi-LSTM	36.9	42.4	42.5	53.5	59.4	57.6
GoogleNet+Transformer	50.0	52.7	47.6	59.9	62.1	58.4
ResNet+Transformer	50.8	53.9	49.3	61.1	63.0	59.9
CLIP-Image+Transformer	51.2	53.6	49.2	61.9	64.0	60.6
CLIP-It: CLIP-Image+Video Caption+Transformer	52.5	54.7	50.0	63.0	65.7	62.8

These datasets are diverse in nature and come with different types of annotations, frame-level scores for TVSum and shot-level scores for SumMe. They are integrated to create the ground-truth using the procedure in [184]. The UT Egocentric dataset consists of 4 videos captured from head-mounted cameras. Each video is about 3-5 hours long, captured in a natural, uncontrolled setting and contains a diverse set of events. The QFVS dataset [139] provides ground-truth generic summaries for these 4 videos. The summaries were constructed by dividing the video into shots and asking 3 users to select the relevant shots. The final ground-truth is an average of annotations from all users.

Note. All the datasets - YouTube [24], Open Video Project (OVP) dataset [112], TVSum [143], SumMe [48], and QFVS [138] were collected by the creators (cited) and consent for any personally identifiable information (PII) was ascertained by the authors where necessary.

Data configuration for TVSum and SumMe. Following previous works [184, 183, 127, 115], we evaluate our approach in three different data settings: Standard, Augment, and Transfer. In the Standard setting, the training and test splits are from the same dataset (i.e. either TVSum or SumMe). For SumMe we use available splits, and for TVSum we randomly select 20% of the videos for testing and construct 5 different splits and report an average result on all 5 splits. For the Augment setting, the training set from one dataset (e.g., TVSum) is combined with all the data from the remaining three datasets (e.g., SumMe, OVP, and YouTube). This setting yields the best performing models due to the additional training data. The Transfer setting is the most challenging of the three. It involves training a model on three datasets and evaluating on the fourth unseen dataset.

Table 3.3: Comparing F1 Scores for generic video summarization on the QFVS dataset.

Supervised	Vid 1	Vid 2	Vid 3	Vid 4	Avg
SubMod [47]	49.51	51.03	64.52	35.82	50.22
QFVS [139]	62.66	46.11	58.85	33.50	50.29
CLIP-Image + bi-LSTM	65.43	56.55	68.63	40.06	57.67
ResNet + Transformer	66.97	58.32	70.10	43.31	59.67
CLIP-Image + Transformer	70.8	61.67	72.43	47.48	63.11
CLIP-It (Gen. Caption)	74.13	63.44	75.86	50.23	65.92
CLIP-It (GT Caption)	84.98	71.26	82.55	61.46	75.06
Unsupervised	Vid 1	Vid 2	Vid 3	Vid 4	Avg
Quasi [188]	53.06	53.80	49.91	22.31	44.77
CLIP-Image + Transformer	65.44	57.21	65.10	41.63	57.35
CLIP-It (Gen. Caption)	67.02	59.48	66.43	44.19	59.28
CLIP-It (GT Caption)	73.90	66.83	75.44	52.31	67.12

Quantitative Results. We compare our method and its ablations to supervised video summarization baselines in Tab.C.6. We report F1 scores on the TVSum and SumMe datasets for all three data settings. Our full method, CLIP-It (CLIP-Image+Video Caption+Transformer), described in Sec. 3.3, outperforms state-of-the-art by a large margin on all three settings. Particularly, in the Transfer setting we outperform the previous state-of-the-art SumGraph [115] by 5% on TVSum and 3% on SumMe, indicating that our model is better than the baselines in generalizing to out-of-distribution data.

To prove the effectiveness of each component of our model, we include comparisons to different ablations. In CLIP-Image+Transformer, we ablate the Language-Guided Attention module and directly pass the CLIP-Image features as input to the transformer. As seen, the performance drops by 2% indicating that conditioning on CLIP language embeddings leads to better summaries. Substituting the Frame-Scoring Transformer with a Bidirectional LSTM in CLIP-Image+bi-LSTM and CLIP-Image+Video Caption+bi-LSTM again results in a performance drop, thus highlighting the need for the Transformer module in our model. For a fair comparison with the baselines, in GoogleNet+Transformer we use the same GoogleNet features provided by Zhang *et al.* [184] and used by all the other baselines and do not include language features. This method still outperforms SumGraph [115]. Replacing the GoogleNet features with ResNet features results in a small performance improvement but CLIP-Image features prove to be most effective.

Tab. C.7 compares F1 scores in the unsupervised setting to other unsupervised baselines. CLIP-It (CLIP-Image+Video Caption+Transformer), outperforms the best performing baseline on all settings on both datasets. We again observe large performance improvements in the Transfer setting and notice that overall our unsupervised method performs almost as well as our supervised



Figure 3.3: Comparison of ground-truth summary to results from CLIP-Image+Transformer and the full CLIP-It model (CLIP-Image+Video Caption+Transformer). The input is a recipe video. Without captions, the model assigns high scores to certain irrelevant frames such as scenes of the woman talking or eating which hurts the precision. With captions, the cross-attention mechanism ensures that frames with important actions and objects are assigned high scores.

counterpart. In CLIP-Image+Transformer we ablate the language component which causes a drop in performance, thus proving that language cues are useful in the unsupervised setting as well. Other ablations yield similar results reiterating the need for each component of our method. We also follow Otani *et al.* [113] and report results on rank based metrics, Kendall’s τ [70] and Spearman’s ρ [196] correlation coefficients in the Supp.

Tab. 3.3 shows F1 scores for the generic setting on the QFVS Dataset [139]. Following [139], we run four rounds of experiments leaving out one video for testing and one for validation, while keeping the remaining two for training. Our method, CLIP-It (Gen. Captions) outperforms both supervised and unsupervised baselines by a large margin on all four videos, and particularly on Video 4, which happens to be the most difficult for all methods. Adding captions helps significantly improve (2%) the summaries and outperforms the CLIP Image + Transformer baseline. To see how well our model would perform if we had perfect captions, we also show results by using the ground-truth captions obtained from VideoSet [175]. Replacing CLIP-Image features with ResNet features causes a drop in performance. Likewise, replacing the Transformer with a bi-LSTM also hurts performance.



Figure 3.4: Qualitative result comparing the generic summary from CLIP-It with the ground-truth summary. The plots showing predicted and ground-truth frame-level scores are similar, indicating that frames that were given a high score in ground-truth were also assigned high scores by our model.

We include results of our method (1) without captions (2) using generated captions (3) using the ground truth captions provided by VideoSet [175] for UT Egocentric and TV Episodes datasets in Tables 3.4 and 3.5. As ground truth, we obtain 15 summaries for each video using the same greedy n-gram matching and ordered subshot selection procedures as previous work [117]. We follow the same procedure as in prior work [175, 117] for creating and evaluating text summaries from video summaries. Our method outperforms [117] in both the supervised and unsupervised settings on both datasets.

Qualitative Results. We highlight the need to use language, specifically dense video captions, for constructing generic video summaries through a qualitative example in Fig. 4.5. The input is a video of a woman demonstrating how to make a chicken sandwich. The ground truth summary shows the scores computed by averaging the annotations from all users as in [184] and the keyframes that received high scores. Next we show the result from the baseline CLIP-Image+Transformer, which uses only visual features and *no language input*. The predicted scores show that the high scoring frames in the ground truth also receive a high score by our baseline, however a lot of irrelevant frames end up receiving a high score too. Thus, when the model finally picks the keyframes it ends up selecting frames where the person is talking or eating the sandwich (shown in red) which do not correspond to the key steps in the video. Adding language guidance via generated captions helps address this problem. The last row shows the captions generated by . The results shown are from our full CLIP-It model. The predicted scores are more similar to ground truth scores and the highest scoring keyframes are the same as the ground truth.

Table 3.4: Results on UT Egocentric dataset [83]

Method	F-Measure	Recall
Supervised		
Submod-V+Both et al. [117]	34.15	31.59
CLIP Image + Transformer	41.58	39.96
CLIP-It : CLIP Image + Gen. Video Caption + Transformer	44.70	43.28
CLIP-It : CLIP Image + GT Video Caption + Transformer	52.10	50.76
Unsupervised		
CLIP Image + Transformer	39.22	37.46
CLIP-It : CLIP Image + Gen. Video Caption + Transformer	42.10	40.65
CLIP-It : CLIP Image + GT Video Caption + Transformer	49.98	47.91

Table 3.5: Results on TV Episodes dataset [83]

Method	F-Measure	Recall
Supervised		
Submod-V+Sem. Rep. et al. [117]	40.90	37.02
CLIP Image + Transformer	47.82	46.02
CLIP-It : CLIP Image + GT Video Caption + Transformer	55.34	53.90
Unsupervised		
CLIP Image + Transformer	45.77	44.01
CLIP-It : CLIP Image + GT Video Caption + Transformer	53.42	52.50

Fig. 4.4 shows compares summaries generated by our method to the ground truth summary on a cooking video from the SumMe dataset. Predicted scores are the frame-wise scores predicted by CLIP-It and GT Scores are the scores computed from user annotations [184]. The top row (True Positives) shows high scoring keyframes which are chosen by both our method and the ground truth, and the green arrows point to the assigned scores. As we see, they are clear, distinct and represent key actions in the recipe. The bottom row (True Negatives) shows frames which are assigned a low score (shown by the red arrows) and are not part of the final summaries (both GT and predicted). E.g., the first frame is irrelevant and corresponds to a segment between key steps, while the second frame has poor lighting and its hard to tell what is being done.



Figure 3.5: Result of our method on the QFVS dataset. The first row shows some frames from the 4 hour long input video. Given the query "book and chair", Summary 1 shows some frames selected by our method. Summary 2 shows frames for the query "Sun and Tree".

Table 3.6: Comparing F1 Scores of different methods on the QFVS dataset.

	Vid 1	Vid 2	Vid 3	Vid 4	Avg
SeqDPP [42]	36.59	43.67	25.26	18.15	30.92
SH-DPP [138]	35.67	42.74	36.51	18.62	33.38
QFVS [139]	48.68	41.66	56.47	29.96	44.19
CLIP-Image + Query + bi-LSTM	54.47	48.59	62.81	38.64	51.13
ResNet + Query + Transformer	55.19	51.03	64.26	39.47	52.49
CLIP-It: CLIP-Image + Query + Transformer	57.13	53.60	66.08	41.41	54.55

Query-Focused Video Summarization

In Query-Focused Video Summarization, the summarization process is conditioned on an input user query, thus, multiple summaries can be obtained for the same video using different input queries.

Dataset. We evaluate our method on the QFVS dataset based on the UT Egocentric videos described earlier. The dataset consists of 46 queries for each of the four videos and user-annotated video summaries for each query.

Quantitative Results. In Tab. 3.6, we compare F1 scores of our method to 3 baselines, SH-DPP [138], Seq-DPP [42], and QFVS [139]. Following [139], we run four rounds of experiments leaving out one video for testing and one for validation, while keeping the remaining two for training. Our full model achieves an avg F1 score of 54.55% outperforming the best baseline (44.19%) by 10%. We would like to point out that our method uses more recent image features compared to the baselines. At the same time, when switching from CLIP to ResNet image embedding, we still improve significantly over the baselines. We expect the improvement to also hold with weaker features (e.g., as demonstrated with GoogleNet results in Tab. C.6, C.7).

Qualitative Results. Fig. 3.5 shows a result on the QFVS dataset for UT Egocentric videos. The input is an egocentric video shot from a head-mounted camera and spans 4 hours. It consists of multiple events from a person’s day, such as reading books, working on the laptop, walking in the streets, and so on. Summary 1 and 2 show results from our CLIP-It method when the input query is “Book and chair” and “Sun and tree” respectively. The frames shown are frames assigned high-scores by our method. As seen, given the same input video, different queries yield different summaries.

3.5 Discussion and Broader Impacts

We introduced CLIP-It, a unified language-guided framework for generic and query focused video summarization. Video summarization is a relevant problem with many use-cases, and our approach provides greater flexibility to the users, allowing them to guide summarization with open-ended natural language queries. We envision potential positive impact from improved user experience if adopted on video platforms such as YouTube. We rely on an off-the-shelf video captioning model [62] and a large-scale vision-language model (CLIP [123]) which may have encoded some inappropriate biases that could propagate to our model. Our visual inspection of the obtained summarization results did not raise any apparent concerns. However, practitioners who wish to use our approach should be mindful of the sources of bias we have outlined above depending on the specific use case they are addressing.

Acknowledgements. We thank Arun Mallya for very helpful discussions and feedback. We’d also like to thank Huijuan Xu for feedback on the draft. This work was supported in part by DoD including DARPA’s XAI, LwLL, and SemaFor programs, as well as BAIR’s industrial alliance programs.

Chapter 4

TL;DW? Summarizing Instructional Videos with Task Relevance & Cross-Modal Saliency

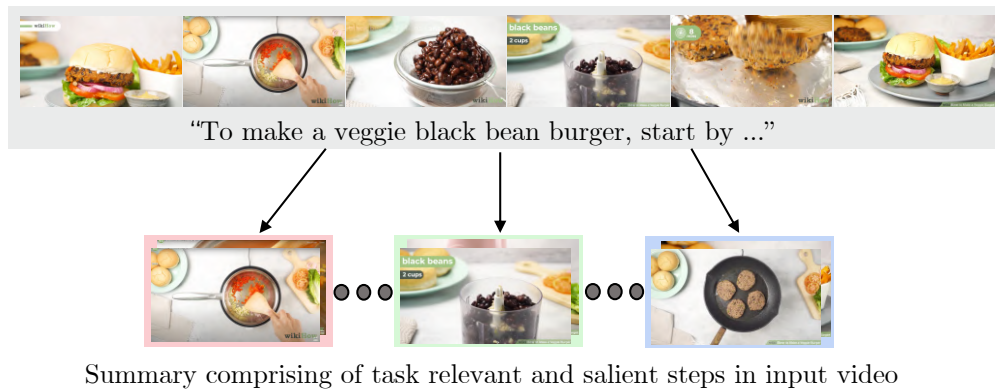


Figure 4.1: **Summarizing Instructional Videos** We introduce an approach for creating short visual summaries comprising steps that are most relevant to the task, as well as salient in the video, i.e. referenced in the speech. For example, given a long video on “*How to make a veggie burger*” shown above, the summary comprises key steps such as *fry ingredients*, *blend beans*, and *fry patty*.

This chapter is based on joint work with Arsha Nagrani, Chen Sun, Miki Rubinstein, Anna Rohrbach, Cordelia Schmid, and Trevor Darrell and is presented much as it appeared in the: ECCV 2022 proceedings.

4.1 Introduction

The search query “*How to make a veggie burger?*” on YouTube yields thousands of videos, each showing a slightly different technique for the same task. It is often time-consuming for a first-time burger maker to sift through this plethora of video content. Imagine instead, if they could watch a compact visual summary of each video which encapsulates all semantically meaningful steps relevant to the task. Such a summary could provide a quick overview of what the longer video has to offer, and may even answer some questions about the task without the viewer having to watch the whole video. In this work, we propose a method to create such succinct visual summaries from long instructional videos.

Since our goal is to summarize videos, we consider prior work on generic [48, 143] and query-focused [139] video summarization. Generic video summarization datasets [48, 143] tend to contain videos from *unrestricted domains* such as sports, news and day-to-day events. Given that annotations are obtained manually, the notion of what constitutes a good summary is subjective, and might differ from one annotator to the next. Query-focused video summarization partially overcomes this subjectivity by allowing users to customize a summary by specifying a natural language query [139, 107]. However, both generic and query-focused approaches require datasets to be annotated manually at a per-frame level. This is very expensive, resulting in very small-scale datasets (25-50 videos) with limited utility and generalization.

Here, we focus on a specific domain – that of instructional videos [152, 195, 104]. We argue that a unique characteristic of these videos is that a summary can be clearly defined as a minimally sufficient *procedural* one, i.e., it must include the steps necessary to complete the task (see Fig. 4.1). To circumvent having to manually annotate our training data, we use an unsupervised algorithm to obtain weak supervision in the form of pseudo ground-truth summaries for a large corpus of instructional videos. We design our unsupervised objectives based on two hypotheses: (i) steps that are relevant to the task will appear across multiple videos of the same task, and (ii) salient steps are more likely to be described by the demonstrator verbally. In practice, we segment the video and group individual segments into steps based on their visual similarity. Then we compare the steps across videos of the same task to obtain *task relevance scores*. We also transcribe the videos using Automatic Speech Recognition (ASR) and compare the video segments to the transcript. We aggregate these *task relevance* and *cross-modal scores* to obtain the *importance scores* for all segments, i.e., our pseudo ground-truth summary.

Next, given an input video and transcribed speech, we train an instructional video summarization network (*IV-Sum*). *IV-Sum* learns to assign scores to short *video segments* using 3D video features which capture temporal context. Our network consists of a video encoder that learns context-aware temporal representations for each segment and a segment scoring transformer (SST) that then assigns importance scores to each segment. Our model is trained end-to-end using the importance scores from the pseudo summaries. Finally, we concatenate the highest scoring segments to form the final video summary.

While we can rely on pseudo ground-truth for training, we collect a clean, manually verified test set to evaluate our method. Since manually creating a labeled test set from scratch would be ex-

tremely expensive, we find a solution in the form of the WikiHow resource¹. WikiHow articles often contain a link to an instructional video and a set of human-annotated steps present in the task along with corresponding images or short clips. To construct our test set (referred to as *WikiHow Summaries*), we automatically localize these images/clips in the video. We obtain localized segments for the images (using a window around the localized frame) and clips, and stitch the segments together to create a summary. This provides us with binary labels for each frame which serve as ground-truth annotations. We evaluate our model on *WikiHow Summaries* and compare it to several baselines and the state-of-the-art video summarization model CLIP-It [107]. Our model surpasses prior work and several baselines on three standard metrics (F-Score, Kendall [70], and Spearman [196] coefficients).

To summarize (pun intended), we introduce an approach for summarizing instructional videos that involves training our *IV-Sum* model on pseudo summaries created from a large corpus of instructional videos. *IV-Sum* learns to rank different segments in the video by learning context-aware temporal representations for each segment and a segment scoring transformer that assigns scores to segments based on their task relevance and cross-modal saliency. Our method is weakly-supervised (it only requires the task labels for videos), multimodal – uses both video and speech transcripts, and is scalable to large online corpora of instructional videos. We collect a high-quality test set, *WikiHow Summaries* for benchmarking instructional video summarization, which will be publicly released. Our model outperforms state-of-the-art video summarization methods on all metrics. Compared to the baselines, our method is especially good at capturing task relevant steps and assigning higher scores to salient frames, as seen through qualitative analysis.

4.2 Related Work

We review several lines of work related to summarization of instructional videos.

Generic Video Summarization. This task involves creating abridged versions of generic videos by stitching together short important clips from the original video [50, 99, 107, 115, 126, 176, 184, 182, 187]. Some of the more recent methods attempt to learn contextual representations to perform video summarization, via attention mechanism [33], graph based [115] or transformer-based [107] methods. Representative datasets include SumMe [48] and TVSum [143], where the ground-truth summaries were created by annotators assigning scores to each frame in the video, which is highly time consuming and expensive. As a consequence, the generic video summarization datasets are small and the quality of the summaries is often very subjective. Here, we focus on instructional videos which contain structure in the form of task steps, thus we have a clear definition of what a good summary should contain - a set of necessary steps for performing that specific task.

Query Focused Video Summarization. To address the subjectivity issues with Generic Summarization, Query Focused Video Summarization allowed for having user defined natural language queries to customize the summaries [66, 139, 165]. A representative dataset is Query Focused Video Summarization [138]; it is very small and the queries correspond to a very narrow set of objects. In contrast, our task is large and we do not rely on any additional user input.

¹<https://www.wikihow.com/>

Step Localization. Step localization (also known as temporal action segmentation) is a related albeit distinct task. It typically implies predicting temporal boundaries of steps when the step labels [124, 152, 195] and even their ordering [11, 14, 27, 60, 78, 125] are given. Representative datasets, COIN [152] and CrossTask [195] consist of instructional videos and a fixed set of steps for each task (from the WikiHow resource), and the task is to localize these steps in the video. Our task is different in that we are only given a video without corresponding input steps. Our model learns to pick out segments that correspond to relevant and salient steps in order to construct a video summary. We discuss and illustrate the shortcomings of the step localization annotations in Sec. 4.5 and Fig. 4.6.

Unsupervised Parsing of Instructional Videos. Closest to ours is the line of work on unsupervised video parsing and segmentation that discovers steps in instructional videos in an unsupervised manner [2, 36, 79, 136, 134]. However, these works - (1) do not focus on video summarization, thus they might miss some salient steps in video, (2) often use very small datasets for training and evaluation that do not capture the broad range of instructional videos found in, e.g., COIN [152] and CrossTask [195].

4.3 Summarizing Instructional Videos

Overview. We propose a novel approach for constructing visual summaries of instructional videos. An instructional video typically consists of a visual demonstration of a specific task, e.g. “*How to make a pancake?*”. Our goal is to construct a visual summary of the input video containing only the steps that are crucial to the task and salient in the video, i.e. referenced in the speech. Fig. 4.2 illustrates an outline of our approach. Our instructional video summarization pipeline consists of two stages - (i) first, we use a weakly supervised algorithm to generate pseudo summaries and frame-wise importance scores for a large corpus of instructional videos, relying only on the task label for each video (ii) next, using the pseudo summaries as supervision, we train an instructional video summarization network which takes as input the video and the corresponding transcribed speech and learns to assign scores to different segments in the input video. The network consists of a video encoder and a segment scoring transformer (SST) and is trained using the importance scores of the pseudo summaries. The final summary is constructed by selecting and concatenating the segments with high importance scores. We first describe our pseudo summary generation algorithm, followed by details on our instructional video summarizer (*IV-Sum*), and the inference procedure.

Generating Pseudo Summaries

Since manually collecting annotations for summarization is expensive and time consuming, we propose an automatic weakly supervised approach for generating summaries that may contain noise but have enough valuable signal for training a summarization network. The main intuition behind our pseudo summary generation pipeline is that given many videos of a task, steps that are crucial to the task are likely to appear across multiple videos (task relevance). Additionally, if a step is important, it is typical for the demonstrator to speak about this step either before, during, or after

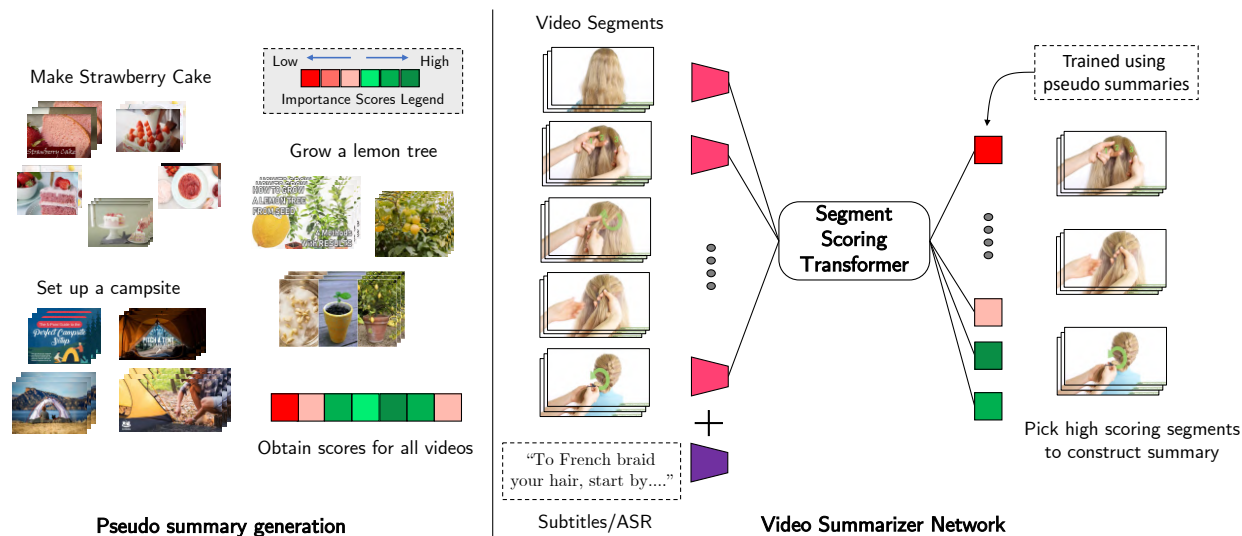


Figure 4.2: **Summarizing Instructional Videos.** We first obtain pseudo summaries for a large collection of videos using our weakly supervised algorithm (more details in Fig. 4.3). Next, using the pseudo summaries as weak-supervision, we train our Instructional Video Summarizer (*IV-Sum*). It takes an input video along with the corresponding ASR transcript and learns to assign importance scores to each segment in the video. The final summary is a compilation of the high scoring video segments.

performing it. Therefore, the subtitles for the video obtained using Automatic Speech Recognition (ASR) will likely reference these key steps (cross-modal saliency). These two hypotheses shape our objectives for generating pseudo summaries.

Task Relevance. We first group videos based on the task. Say videos $V_i, i \in [1, \dots, \mathcal{K}]$ are \mathcal{K} videos from the same task, as shown in Fig. 4.3. For a given video, we divide it into \mathcal{N} equally sized non-overlapping segments $s_i, i \in [1, \dots, \mathcal{N}]$ and embed each segment using a pre-trained 3D CNN video encoder g_{vid} [103]. We merge segments along the time axis based on their dot-product similarity, i.e. if similarity of a segment to the one prior to it is greater than a threshold, the two are grouped together and the joint feature representation is an average of the feature representation of the two segments. The threshold for similarity is heuristically set to be 90% of the maximum similarity between any two segments in the video. We call these merged segments *steps*, as they typically correspond to semantic steps as we show through qualitative results in supplemental. We do this for all \mathcal{K} videos in the task, and then compare each step to all the \mathcal{S} steps across all \mathcal{K} videos of the task. We assign *task relevance scores* trs_{S_i} , to each step $S_i, i \in \mathcal{S}$ based on its visual similarity to all the \mathcal{S} steps from all \mathcal{K} videos of this task, as shown below:

$$\text{trs}_{S_i} = \frac{1}{|\mathcal{S}|} \sum_{j \in \mathcal{S}} g_{vid}(S_i) \cdot g_{vid}(S_j)$$

Cross-Modal Saliency. We also compare each video step to each sentence in the transcript of the

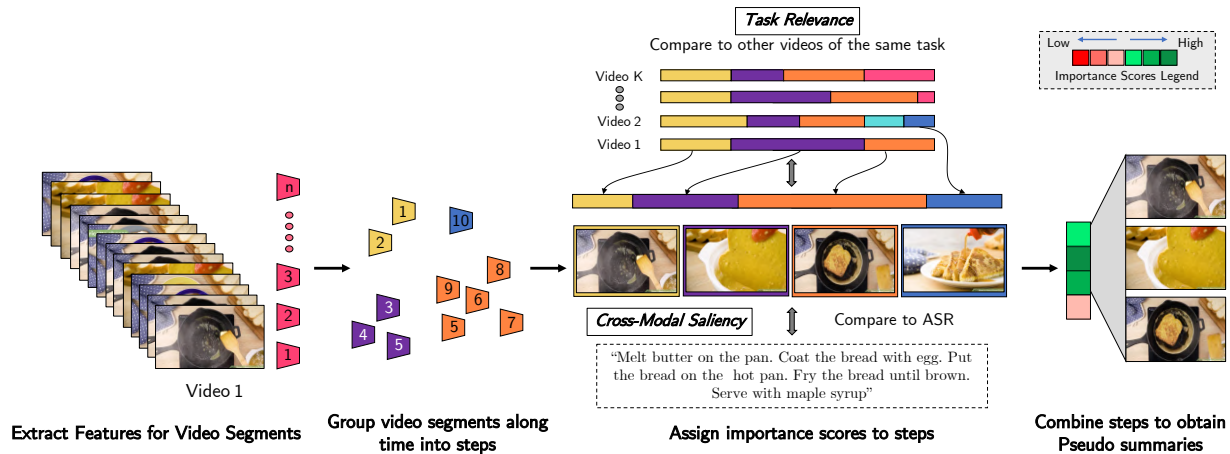


Figure 4.3: **Pseudo Summary Generation.** To generate the pseudo summary, we first uniformly partition the video into segments, then group the segments based on visual similarity into steps (shown in different colors), assign *importance scores* to steps based on *Task Relevance* and *Cross-Modal Saliency*, and then pick high scoring steps to obtain pseudo summaries.

same video. This enforces our idea that if a step is important, it will likely be referenced in the speech. To do this, we encode both, the input segments and the transcript sentences, using a pre-trained video-text model where the video and text streams are trained jointly using MIL-NCE loss [103]. Each visual step is assigned a *cross-modal score* by averaging its similarity over all the sentences.

Each step (and all the segments in it) is then assigned an importance score that is an average of the *task relevance* and the *cross-modal scores*. This constitutes our pseudo summary scores. For any given video, the top $t\%$ highest scoring steps are retained to be a part of the summary.

Instructional Video Summarizer (*IV-Sum*)

Recall that our goal is to construct a visual summary of any instructional video by picking out the important steps in it, without having to rely on other videos of the same task or the task label. To do this, we use the pseudo summaries generated above as weak supervision to train *IV-Sum*, which learns to assign importance scores to individual segments in the video using only the information in the video and the corresponding transcripts as seen in Fig. 4.2. While some prior summarization methods operate on independent frames [107, 115], *IV-Sum* operates on non-overlapping segments $s_i, i \in [1, \dots, \mathcal{N}]$, and learns *context-aware temporal representations* using a 3D CNN video encoder f_{vid} . The transcript is projected onto the same embedding space using a text encoder f_{text} , and the text representations are concatenated individually to each of the segments. To contextualize information across several segments, we use a segment scoring encoder-only transformer [156] f_{trans} with positional embeddings, that assigns importance scores Y'_{s_i} to each segment as shown in Eq. 4.1. The network is trained using supervision from the importance scores of the pseudo summaries Y_{s_i} , using Mean-Squared Error Loss as shown in Eq. 4.2.

$$Y'_{s_i} = f_{\text{trans}}(\text{concat}(f_{\text{text}}(\text{transcript}), f_{\text{vid}}(s_i))) \quad \forall i \in \mathcal{N} \quad (4.1)$$

$$\mathcal{L}_{\text{IV-Sum}} = \sum_{i \in \mathcal{N}} \text{MSE}(Y'_{s_i}, Y_{s_i}) \quad (4.2)$$

During inference, we sort the segments based on the predicted scores and assign the label 1 to the top $t\%$ of the segments, and the label 0 to the remaining ones. When a segment is assigned a label, all the frames in the segment also get assigned the same label. The summary is constructed by stitching together all the frames with label 1.

4.4 Instructional Video Summarization Datasets

We describe the details of the data collection process for the annotations used in our work — *Pseudo Summaries* annotations for training and the *WikiHow Summaries* annotations for evaluation.

Pseudo Summaries Training Dataset. As described in Sec. 4.3, we use the pseudo summary generation process for creating our training set. We use the videos and task annotations from COIN [152] and CrossTask [195] datasets for creating our training datasets.

COIN: COIN consists of 11K videos related to 180 tasks. As this is a dynamic YouTube dataset, we were able to obtain 8,521 videos at the time of this work.

Cross-Task: CrossTask consists of 4,700 instructional videos (of which we were able to access 3,675 videos) across 83 different tasks.

Pseudo Summaries: We combined the two datasets to create pseudo summaries comprising of 12,160 videos, whilst using the videos that were common to both datasets only once. They span 263 different tasks, have an average length of 3.09 minutes, and in total comprise of 628.53 hours of content. The summary videos that were constructed using our pseudo ground-truth generation pipeline are 1.71 minutes long on an average, with each summary being 60% of the original video. While it is possible to construct pseudo summaries using the step-localization annotations, we show in Sec. 4.5 that such summaries may miss important steps or do not pick up on steps that are salient in the video. Moreover, our pseudo summary generation mechanism is weakly-supervised, requiring only task annotations and no step-localization annotations.

WikiHow Summaries Dataset. To provide a test bed for instructional video summarization, we automatically create and manually verify *WikiHow Summaries*, a video summarization dataset consisting of 2,106 input videos and summaries, where each video describes a unique task. Each article on the WikiHow Videos website consists of a main instructional video demonstrating a task that often includes promotional content, clips of the instructor speaking to the camera with no visual information of the task, and steps that are not crucial for performing the task. Viewers who want an overview of the task would prefer a shorter video without all of the aforementioned irrelevant information. The WikiHow articles (e.g., see How to Make Sushi Rice) contain exactly this: corresponding text that contains all the important steps in the video listed with accompanying images/clips illustrating the various steps in the task. These manually annotated articles are a good source for automatically creating ground-truth summaries for the main videos. We obtain the

Table 4.1: **Instructional Video Summarization Datasets Statistics.** † Our *WikiHow Summaries* dataset was created automatically using a scalable pipeline, but manually verified for correctness.

	TVSum	SumMe	Pseudo Summaries	WikiHow Summaries
Number of videos	50	25	12160	2106
Annotation	Manual	Manual	Automatic	Manually verified†
Number of Tasks/Categories	10	25	185	20
Total Input Duration (Hours)	3.5	1.0	628.53	42.94

summaries and the corresponding labels and importance scores using the following process (see supp. for an overview figure):

1. Scraping WikiHow videos. We scrape the WikiHow Videos website for all the long instructional videos along with each step and the images/video clips (GIFs) associated with the step.

2. Localizing images/clips. We automatically localize these images/clips in the main video by finding the closest match in the video. To localize an image, we compare ResNet50 [51] features of the image and to that of all the frames in the video. The most similar frame is selected and this step is localized in the input video to a 5 second window centered around the frame. If the step contains a video clip/GIF, we localize the first frame of the video clip/GIF in the input video by similarly comparing ResNet features, as above, and the localization is set to be the length of the step video clip.

3. Ground-truth summary from localized clips. We stitch the shorter localized clips together to create the ground truth summary video. Consequently, we assign labels to each frame in the input video, depending on whether it belongs to the input summary (label 1) or not (label 0). To obtain importance scores, we partition each input video into equally sized segments (same as in Sec. 4.3) and compute the importance score for each segment to be the average of the labels assigned to the individual frames in the segment.

4. Manual verification. We verified that the summaries are at least 30% of the original video and manually fixed summaries that were extremely short/long.

Online Longevity and Scalability. We note that a common problem plaguing YouTube datasets today is shrinkage of datasets as user uploaded videos are taken down by users (eg. Kinetics [13]). WikiHow articles are less likely to be taken down, and this is an actively growing resource as new How-To videos are released and added (25% growth since we collected the data). Hence there is a potential to continually increase the size of the dataset.

For each video, we provide the following: (i) frame-level binary labels (ii) the summary formed by combining the frames with label 1 (iii) segment-level importance scores between 0 and 1, which are computed as an average of the importance scores for all the frames in the segment (iv) the localization of the visual steps in the video (i.e. the frames associated with each step). We also scrape natural language descriptions of each step as a bonus that could be useful for future work. We divide our WikiHow dataset into 768 validation and 1,339 test videos. Tab. 4.1 shows the statistics of both our datasets. Both datasets are much larger in size compared to existing generic video

summarization datasets, contain a broader range of tasks, and are scalable.

4.5 Experiments

Next, we describe the experimental setup and evaluation for instructional video summarization. We compare our method to several baselines, including CLIP-It [107], the state-of-the-art on generic and query-focused video summarization.

Implementation Details. For the video and text encoders, we use an S3D [168] network, initialized with weights from pre-training on HowTo100M [104] using the MIL-NCE loss [103]. We fine-tune the *mixed_5** layers and freeze the rest. The segment scoring transformer is an encoder consisting of 24 layers and 8 heads and is initialized randomly. The network is trained using the Adam optimizer [72], with learning rate of 0.01, and a batch size of 24. We use Distributed Data Parallel to train for 300 epochs across 8 NVIDIA RTX 2080 GPUs. Additional implementation details are mentioned in supplemental.

Metrics. To evaluate instructional video summaries, we follow the evaluation protocol used in past video summarization works [183, 115, 107] and report Precision, Recall and F-Score values. As described in Sec. 4.4, each video in the *WikiHow Summaries* dataset contains the ground-truth labels Y_l (binary labels for each frame in the video) and the ground-truth scores Y_s (importance scores in the range [0-1] for each segment in the video). We compare the binary labels predicted for the frames in the video Y_l' , to the ground truth labels Y_l , and measure F-Score, Precision and Recall, as defined in prior summarization works [127, 126].

While these scores assess the quality of the predicted frame-wise binary labels, to assess the quality of the predicted segment-wise importance scores Y_s' , we follow Otani *et al.* [113], and report results on the rank-based metrics Kendall’s τ [70] and Spearman’s ρ [196] correlation coefficients. We first rank the video frames according to the generated importance scores Y_s' and the ground-truth importance scores Y_s . We then compare the generated ranking to each ground-truth ranking of video segments for each video obtained from the frame-wise binary labels as described in Sec. 4.4. The final correlation score is computed by averaging over the individual scores for each video.

Baselines. We compare our method to the state-of-the-art video summarization model CLIP-It [107]. To validate the need for pseudo summaries, we construct three unsupervised baselines as alternatives to our pseudo summary generation algorithm. We first describe the three unsupervised baselines.

Frame Cross-Modal Similarity. We sample frames (at the same FPS used by our method) from an input video and compute the similarity between CLIP (ViT-B/32) [123] frame embeddings and CLIP text embeddings of each sentence in the transcript. The embeddings do not encode temporal information but leverage the priors learned by the CLIP model. Based on the scores assigned to each frame, we threshold $t\%$ of the higher scoring frames to be part of the summary. Frame scores are propagated to the segments they belong to, and the summary is a compilation of the chosen segments.

Segment Cross-Modal Similarity. We uniformly divide the video into segments and compute MIL-NCE [103] video features for each segment. We embed each sentence in the transcript to the same feature space using the MIL-NCE text encoder. We compute the pairwise similarity between

Table 4.2: **Instructional Video Summarization results on WikiHow Summaries.** We compare F-Score, Kendall and Spearman correlation metrics of our method IV-Sum, to all the baselines. Our method achieves state-of-the-art on all three metrics.

Method				F-Score		τ [70]	ρ [196]
				Val	Test	Test	Test
	ASR	RGB	Pseudo				
Frame Cross-Modal Similarity	✓	✓	-	52.8	53.1	0.022	0.051
Segment Cross-Modal Similarity	✓	✓	-	55.1	55.5	0.034	0.060
Step Cross-Modal Similarity	✓	✓	-	57.9	58.3	0.037	0.061
CLIP-It with captions [107]	-	✓	-	22.5	22.1	0.036	0.064
CLIP-It with ASR [107]	✓	✓	-	27.9	27.2	0.055	0.088
CLIP-It with ASR	✓	✓	✓	62.5	61.8	0.093	0.191
IV-Sum without ASR	-	✓	✓	65.8	65.2	0.095	0.202
IV-Sum	✓	✓	✓	67.9	67.3	0.101	0.212

all video segments and the sentences, and average over sentences to obtain a score for each segment. Our intuition is that since demonstrators typically describe the important steps shortly before, after or while performing them, a high similarity between the visuals and transcripts would directly correlate with the significance of the step. We filter $t\%$ of the highest scoring segments, where t is determined heuristically using the *WikiHow Summaries* validation set and is consistent across all baselines and our model. The filtered segments are stitched together to form the summary.

Step Cross-Modal Similarity. We first group segments into steps and then compare them to the ASR transcripts. For this we employ the technique described in Sec. 4.3, i.e. we extract MIL-NCE features for the video segments and group them together based on their similarity to form steps.² The embedding for a step is set to be the average of all the segment embeddings in it. If a step is similar to the transcript, all the segments in that step are chosen to be part of the summary. This baseline is the closest to our pseudo summary generation algorithm.

Next, we describe the CLIP-It baseline and ablations, trained with supervision.

CLIP-It with captions. We evaluate CLIP-It [107] trained on TVSum [143], SumMe [48], OVP [112], and YouTube [24] against our *WikiHow Summaries*. We use the same protocol as in CLIP-It for evaluation and describe further details in supplemental. For language-conditioning, we follow CLIP-It and generate captions for the *WikiHow Summaries* dataset using BMT [62]; we feed these as input to the CLIP-It model.

CLIP-It with ASR transcripts. We evaluate the same CLIP-It model above by replacing captions with ASR transcripts, so as to allow for a fair comparison with the baselines and our method, IV-Sum which use ASR transcripts.

²Since we process a single input video (not multiple videos per task), we can not use the Task Relevance component.

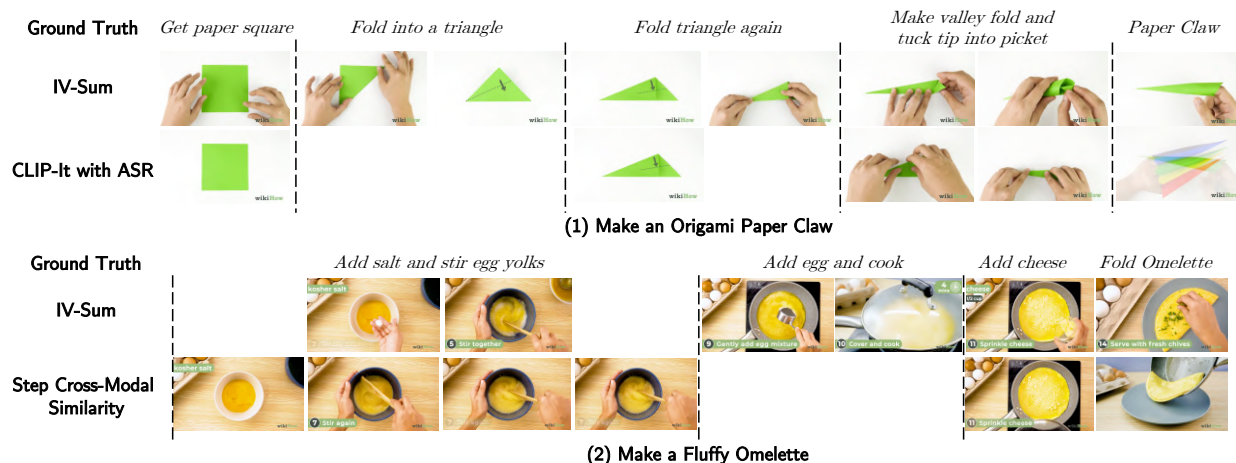


Figure 4.4: **Qualitative comparisons to baselines.** We show the steps in the ground-truth as text (note we never train with step descriptions, these are shown here simply for illustrative purposes) and compare frames selected in summaries generated by our method IV-Sum, CLIP-It with ASR, and Step Cross-Modal Similarity. In (1), CLIP-It misses steps which are deemed important by our method (“*Fold into a triangle*”) and assigns higher scores to less salient frames for the step (“*Make valley fold and tuck tip into picket*”) where neither the valley fold nor the picket are clearly visible. In (2), Step Cross-Modal Similarity misses (“*Add egg and cook*”) and selects too many redundant frames for the step (“*Add salt and stir egg yolks*”).

CLIP-It with ASR transcripts trained on Pseudo Summaries. We train CLIP-It from scratch on our Pseudo-GT Summaries dataset using ASR transcripts from the videos in place of captions.

Quantitative Results. We compare the baselines to the two versions of IV-Sum, one with ASR transcripts and another without. To train IV-Sum without transcripts, we simply eliminate the text encoder (f_{text}) in Eq. 4.1 and pass only the visual embeddings of the individual segments to the transformer. We report F-Score, Kendall’s τ and Spearman’s ρ coefficients in Tab. C.6. As seen, IV-Sum (both with and without ASR transcripts), outperforms all the baselines on all metrics. Particularly, we achieve notable improvements on the correlation metrics that compare the saliency scores, attesting to our model’s capabilities to assign higher scores to segments that are more relevant. We also observe that CLIP-It trained using the pseudo summaries generated by our method has a strong boost in performance compared to CLIP-It trained on generic video summarization datasets, reinforcing the effectiveness of our pseudo summaries for training. The best method among the unsupervised ones is Step Cross-Modal Similarity, a “reduced” version of our pseudo summary generation method.

Qualitative Results. We present qualitative results in Fig. 4.4. We show frames in the summaries generated by our method IV-Sum, CLIP-It with ASR transcripts (trained on generic video summarization datasets), and Step Cross-Modal Similarity. We also list the steps in the ground-truth as text (for illustrative purposes). In Fig. 4.4 (1), CLIP-It misses the step “*Fold into a triangle*”, as it

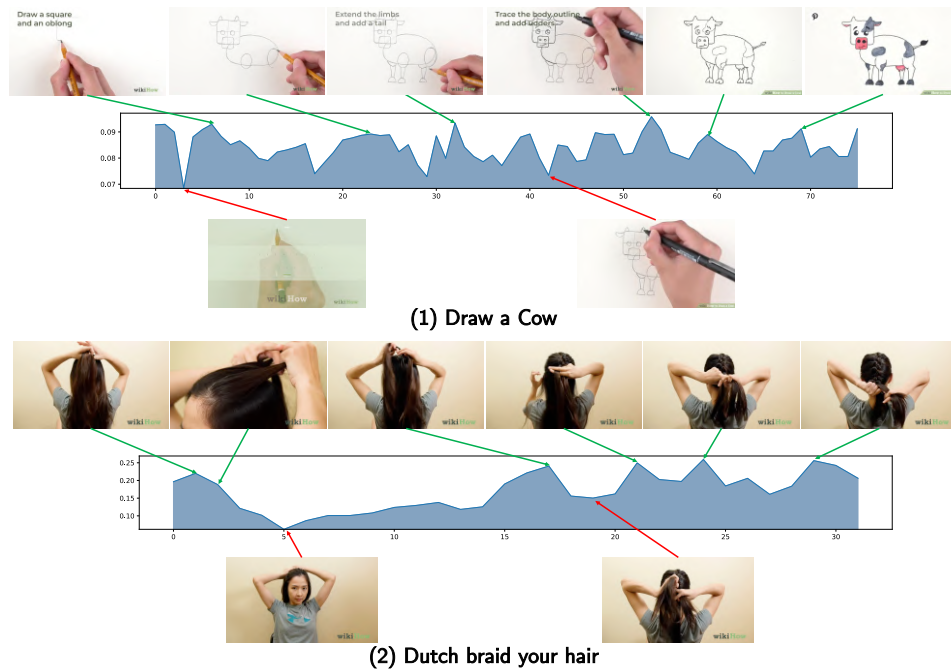


Figure 4.5: **Qualitative results.** We show summaries from our method IV-Sum along with the predicted importance scores. The **green** and **red** arrows point to frames that were assigned a high and low scores, respectively. Our model correctly assigns higher scores to frames from all the steps that are relevant and lower scores to frames which aren’t crucial to the task (as in (1)) and frames which don’t belong to a step (as in (2)).

optimizes for diversity among the frames and was trained on a small dataset that does not generalize well to our domain. It also picks the less salient frames for the step “*Make valley fold and tuck tip into picket*”, whereas our model correctly identifies all the steps and assigns higher scores to the more salient frames. The summary from the Step Cross-Modal Similarity baseline, shown in Fig. 4.4 (2), assigns high scores to several redundant frames (“*Add salt and stir egg yolks*”), but misses “*Add egg and cook*”.

Fig. 4.5 shows results from our method along with the predicted frame-wise importance scores. The **green** and **red** arrows point to frames that are assigned the highest and lowest scores by our method, respectively. As seen, our method assigns high scores to frames in task relevant and salient steps and low scores to frames which aren’t crucial to the step, like in Fig. 4.5 (1), or do not belong to a step, like in Fig. 4.5 (2) where the person is talking to the camera.

Ablations. We compare different approaches to generate pseudo summaries for training our instructional video summarizer network – (i) First, we ablate the two objectives, Task Relevance and Cross-Modal Saliency, used to generate the pseudo summaries. (ii) Next, we replace the annotations from our pseudo summary generation pipeline with step localization annotations. We include model and loss ablations in the supplemental.

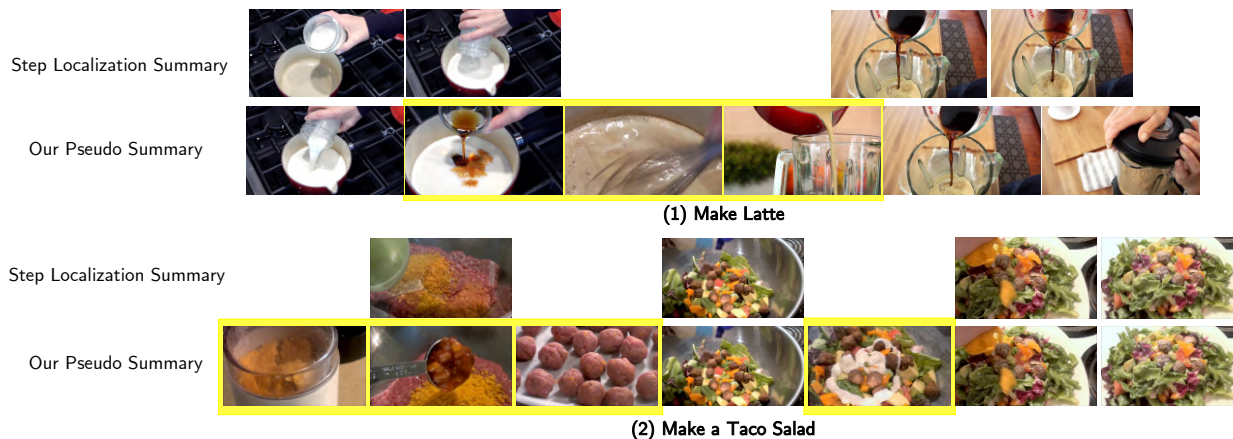


Figure 4.6: **Pseudo summaries vs step-localization annotations.** We compare frames in our automated pseudo summary to the step localization manual annotations, aligned temporally. Frames corresponding to steps that are identified by our method but missed by step localization are highlighted in yellow.

Table 4.3: **Pseudo Summary Variations.** We report results on two variations of generating the pseudo summaries: (i) ablating the objectives (ii) using step localization annotations to generate pseudo summaries.

(a) **Ablating objectives.** We ablate the two objectives in our pseudo summary generation pipeline.

Method	F-Score
Task-Consistency only	64.1
Cross-Modal Similarity only	61.0
Both	67.9

(b) **Using Step-Localization Annotations.** We compare pseudo summaries from step-localization annotations with our approach.

Method	F-Score
IV-Sum (Step Localization)	57.6
IV-Sum (Ours)	66.8

(i) *Ablating Objectives.* We ablate the two objectives, Task Relevance and Cross-Modal Saliency, used for generating pseudo summaries, in Tab. 4.3a. We train IV-Sum on different versions of pseudo summaries and report F-Scores on the *WikiHow Summaries* validation set. Combining both objectives is more effective than using each objective individually.

(ii) *Using Step Localization Annotations.* COIN and CrossTask datasets contain temporal localization annotations of a generic set of steps pertaining to the task in the videos. We use these annotations to extract the visual segments corresponding to the steps and concatenate them to form a summary. We assign binary labels to each frame, depending on whether they belong in the summary or not. We then use these step-localization summaries as supervision to train our model, IV-Sum with a weighted-CE loss [107] as this works best for binary labels. In Tab. 4.3b, we compare this to IV-Sum trained on pseudo summaries generated using our pipeline and report F-Scores

on our *WikiHow Summaries* validation set. As seen, IV-Sum trained on our generated summaries outperforms IV-Sum trained using step-localization summaries. We qualitatively compare our automatic pseudo summaries to the manually labeled step localization annotations in Fig. 4.6. Often the step annotations only cover a few steps and miss other crucial steps as shown in yellow in (1). In (2), we observe that our pseudo summary retrieves steps that are unique to the task which the step localization annotation doesn't include.

4.6 Conclusion

We introduce a novel approach for generating visual summaries of instructional videos — a practical task with broad applications. Specifically, we overcome the need to manually label data in two important ways. For training, we propose a weakly-supervised method to create pseudo summaries for a large number of instructional videos. For evaluation, we leverage WikiHow (its videos and step illustrations) to automatically build a *WikiHow Summaries* dataset. We manually verify that the obtained summaries are of high quality. We also propose an effective model to tackle instructional video summarization, IV-Sum, that uses temporal 3D CNN representations, unlike most prior work that relies on frame-level representations. We demonstrate that all components of the proposed approach are effective in a comprehensive ablation study.

Acknowledgements: We thank Daniel Fried and Bryan Seybold for valuable discussions and feedback on the draft. This work was supported in part by DoD including DARPA's LwLL, PTG and/or SemaFor programs, as well as BAIR's industrial alliance programs.

Chapter 5

Learning and Verification of Task Structure in Instructional Videos

5.1 Introduction

Picture this, you're trying to build a bookshelf by watching a YouTube video with several intricate steps. You're annoyed by the need to repeatedly hit pause on the video and you're unsure if you have gotten all the steps right so far. Fortunately, you have an interactive assistant that can guide you through the task at your own pace, verifying each step as you perform it and interrupting you if you make a mistake. A composite task such as “*making a bookshelf*” involves multiple fine-grained activities such as “*drilling holes*” and “*adding support blocks*.” Accurately categorizing these activities requires not only recognizing the individual steps that compose the task but also understanding the task structure, which includes the temporal ordering of the steps and multiple plausible ways of executing a step (e.g., one can beat eggs with a fork or a whisk). An ideal interactive assistant has both a high-level understanding of a broad range of tasks, as well as a low-level understanding of the intricate steps in the tasks, their temporal ordering, and the multiple ways of performing them.

As seen in Fig. 6.1, prior work [84, 93] models step representations of a single step independent of the overall task context. This might not be the best strategy, given that steps for a task are related, and the way a step is situated in an overall task may contain important information about the step. To address this, we pre-train our model with a masked modeling objective that encourages the step representations to capture the *global context* of the entire video. Prior work lacks a benchmark for detecting mistakes in videos, which is a crucial component of verifying the quality of instructional video representations. We introduce a mistake detection task and dataset for verifying if the task in a video is executed correctly—i.e. if each step is executed correctly and in the right order.

This chapter is based on joint work with Licheng Yu, Ning Zhang, Sean Bell, and Trevor Darrell. More details are on the project webpage: https://medhini.github.io/task_structure/.

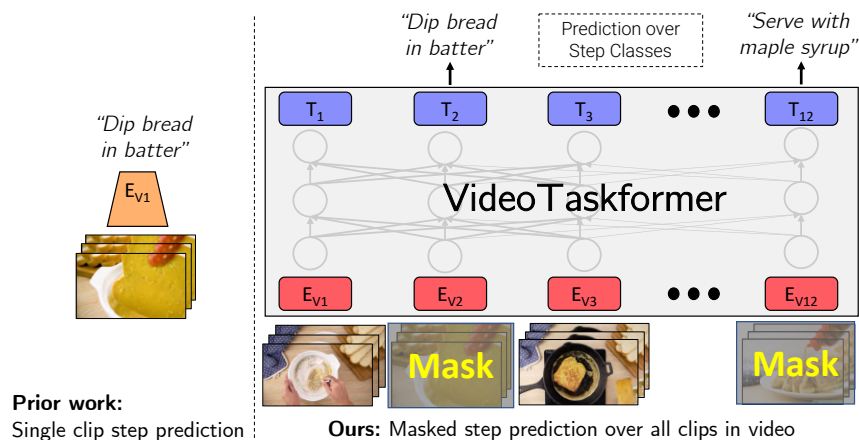


Figure 5.1: Prior work [93, 84] learns step representations from single short video clips, independent of the task, thus lacking knowledge of task structure. Our model, VideoTaskformer, learns step representations for masked video steps through the global context of all surrounding steps in the video, making our learned representations aware of task semantics and structure.

Our goal is to learn representations for the steps in the instructional video which capture semantics of the task being performed such that each step representation contains information about the surrounding context (other steps in the task). To this end, we train a model VideoTaskformer, using a masked step pre-training approach for learning step representations in instructional videos. We learn step representations jointly for a whole video, by feeding multiple steps to a transformer, and masking out a subset. The network learns to predict labels for the masked steps given just the visual representations of the remaining steps. The learned contextual representations improve performance on downstream tasks such as forecasting steps, classifying steps, and recognizing procedures.

Our approach of modeling steps further enables a new method for mistake identification. Recall, our original goal was to assist a user following an instructional video. We synthetically generate a mistakes dataset for evaluation using the step annotations in COIN [152]. We consider two mistake types: mistakes in the steps of a task, and mistakes in the ordering of the steps of a task. For the first, we randomly replace the steps in a video with steps from a similar video. For the second, we re-order the steps in a task. We show that our network is capable of detecting both mistake types and outperforms prior methods on these tasks.

Additionally, we evaluate representations learned by VideoTaskformer on three existing benchmarks: step classification, step forecasting, and procedural activity recognition on the COIN dataset. Our experiments show that learning step representation through masking pre-training objectives improves the performance on the downstream tasks. We will release code, models, and the mistake detection dataset and benchmark to the community.

5.2 Related Works

Instructional Video Datasets and Tasks. Large-scale narrated instructional video datasets [22, 104, 152, 191, 195] have paved the way for learning joint video-language representations and task structure from videos. More recently, datasets such as Assembly-101 dataset [135] and Ikea ASM [8] provide videos of people assembling and disassembling toys and furniture. Assembly-101 also contains annotations for detecting mistakes in the video. Some existing benchmarks for evaluating representations learned on instructional video datasets include step localization in videos [22, 152], step classification [22, 152, 195], procedural activity recognition [152], and step forecasting [93]. In our work, we focus on a broad range of instructional videos found in HowTo100M [104] and evaluate the learned representations on the downstream tasks in COIN [152] dataset. We additionally introduce 3 new benchmarks for detecting mistakes in instructional videos and forecasting long-term activities.

Procedure Learning from Instructional Videos. Recent works have attempted to learn procedures from instructional videos [7, 15, 93, 121, 159]. Most notably, [15] generates a sequence of actions given a start and a goal image. [7] finds temporal correspondences between key steps across multiple videos while [121] distinguishes pairs of videos performing the same sequence of actions from negative ones. [93] uses distant supervision from WikiHow to localize steps in instructional videos. Contrary to prior works, our step representations are aware of the task structure as we learn representations globally for all steps in a video jointly, as opposed to locally, as done in past works.

Video Representation Learning. There has been significant improvement in video action recognition models over the last few years [4, 34, 35, 95]. All of the above methods look at trimmed videos and focus on learning short-range atomic actions. In this work, we build a model that can learn longer and more complex actions, or steps, composed of multiple short-range actions. For example, the first step in Fig. 6.1, “*Make batter*”, is composed of several atomic actions such as “*pour flour*” and “*whisk*”. There have also been works [93, 103, 122, 147, 171] which learn representations for longer video clips containing semantically more complex actions. Our work falls into this line of work.

5.3 Learning Task Structure through Masked Modeling of Steps

Our goal is to learn task-aware step representations from a large corpus of instructional videos. To this end, we develop VideoTaskformer, a video model pre-trained using a BERT [26] style masked modeling loss. In contrast to BERT and VideoBERT [147], we perform masking at the step level, which encourages the network to learn step embeddings that encapsulate the semantics and temporal ordering of steps within the task.

Our framework consists of two steps: pre-training and fine-tuning. During pre-training, VideoTaskformer is trained on weakly labeled data on the pre-training task. For fine-tuning, VideoTaskformer is first initialized with the pre-trained parameters, and a subset of the parameters is fine-tuned

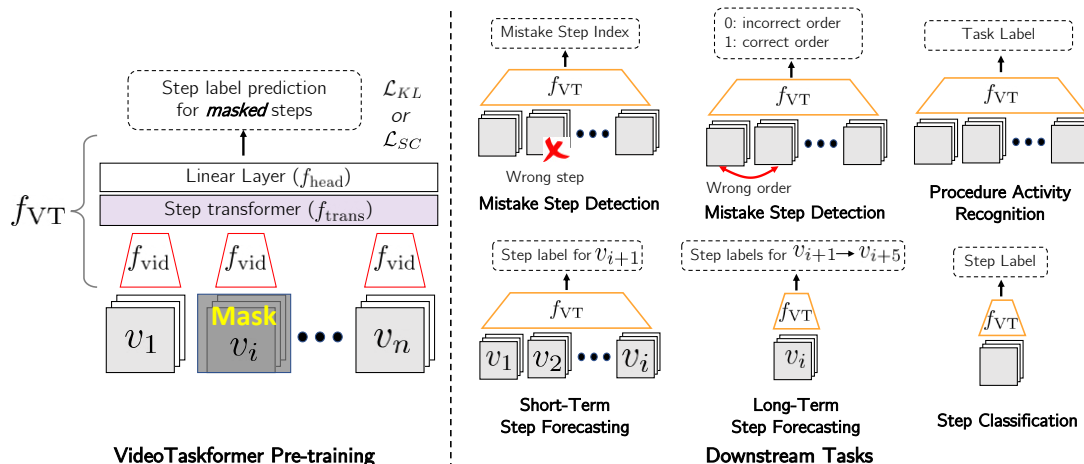


Figure 5.2: **VideoTaskformer Pre-training (Left)**. VideoTaskformer f_{VT} learns step representations for the masked out video clip v_i , while attending to the other clips in the video. It consists of a video encoder f_{vid} , a step transformer f_{trans} , and a linear layer f_{head} , and is trained using weakly supervised step labels. **Downstream Tasks (Right)**. We evaluate step representations learned from VideoTaskformer on 6 downstream tasks.

using labeled data from the downstream tasks. Each downstream task yields a separate fine-tuned model.

We first provide an overview of the pre-training approach before delving into details of the individual components.

Overview. Our approach for pre-training VideoTaskformer is outlined in Fig. 5.2. Consider an instructional video V consisting of K video clips $v_i, i \in [1, \dots, K]$ corresponding to K steps in the video. A step $v_i \in \mathbb{R}^{L \times H \times W \times 3}$ is a sequence of L consecutive frames depicting a step, or semantic component of the task. For example, for the task “*Making a french toast*”, examples of steps include “*Whisk the batter*”, and “*Dip bread in batter.*” We train a video model VideoTaskformer f_{VT} to learn step representations. We mask out a few clips in the input V and feed it to f_{VT} which learns to predict step labels for the masked-out clips. We evaluate the embeddings learned by our pre-training objective on 6 downstream tasks: step classification, procedural activity recognition, step forecasting, mistake step detection, mistake ordering detection, and long term forecasting.

Below, we provide more details on how we pre-train VideoTaskformer using a masked step modeling loss, followed by fine-tuning details on the downstream tasks.

Pre-training VideoTaskformer with Masked Step Modeling

We extend masked language modeling techniques used in BERT and VideoBERT to learn step representations for instructional videos. While BERT and VideoBERT operate on language and visual tokens respectively, VideoTaskformer operates on clips corresponding to steps in an instructional

video. By predicting weakly supervised natural language step labels for masked out clips in the input video, VideoTaskformer learns semantics and long-range temporal interactions between the steps in a task. Unlike prior works wherein step representations are learned from local short video snippets corresponding to the step, our step representations are from the entire video with all the steps as input and capture *global context* of the video.

Masked Step Modeling. Let $V = \{v_1, \dots, v_K\}$ denote the visual clips corresponding to K steps in video V . The goal of our Masked Step Modeling pre-training setup is to encourage VideoTaskformer to learn representations of clips v_i that are aware of the semantics of the corresponding step and the context of the surrounding task. To this end, the task for pre-training is to predict categorical natural language step labels for the masked out steps. While we do not have ground truth step labels, we use the weak supervision procedure proposed by [93] to map each clip v_i to a distribution over step labels $p(y_i | v_i)$ by leveraging the noisy ASR annotations associated with each clip. The distribution $p(y_i | v_i)$ is a categorical distribution over a finite set of step labels Y . More details are provided in Sec. 5.3.

Let $M \subseteq [1, \dots, K]$ denote some subset of clip indices (where each index is included in M with some masking probability r , a hyperparameter). Let $V_{\setminus M}$ denote a partially masked-out sequence of clips: the same sequence as V except with clips v_i masked out for all $i \in M$.

Let f_{VT} represent our VideoTaskformer model with parameters θ . f_{VT} is composed of a video encoder model f_{vid} which encodes each clip v_i independently, followed by a step transformer f_{trans} operating over the sequence of clip representations, and finally a linear layer f_{head} (which includes a softmax). The input to the model is an entire video (of size $K \times L \times H \times W \times 3$) and the output is of size $K \times S$ (where S is the output dimension of the linear layer).

We pre-train f_{VT} by inputting a masked video $V_{\setminus M}$ and predicting step labels y_i for each masked-out clip v_i , as described below. For the downstream tasks, we extract step-aware representations using f_{VT} by feeding an unmasked video V to the model. We then extract the intermediate outputs of f_{trans} (which are of size $K \times D$, where D is the output embedding size).

To predict step labels for masked-out steps at pre-training time, we consider two training objectives: (1) step classification, and (2) distribution matching. We describe them below in the context of Masked Step Modeling.

Step classification loss. We use the outputs of f_{VT} to represent an S -dimensional prediction distribution over steps, where $S = |Y|$. We form the target distribution by placing all probability mass on the best textual step description y_i^* for each clip v_i according to the weak supervision process. That is,

$$y_i^* = \operatorname{argmax}_{y \in Y} p(y | v_i). \quad (5.1)$$

We calculate the cross entropy between the predicted and target distributions for each masked out clip, yielding the following expression:

$$-\log([f_{\text{VT}}(V_{\setminus M})]_j) \quad (5.2)$$

where j is the index of y_i^* in Y , i.e., such that $y_i^* = Y_j$. To get the final training objective for a single masked video $V_{\setminus M}$, we sum over all indices $i \in M$, and minimize with respect to θ .

Distribution matching loss. For this objective, we treat the distribution of step labels $p(y_i | v_i)$ from weak supervision as the target distribution for each clip v_i . We then compute the KL Divergence between the prediction distribution $f_{\text{VT}}(V_{\setminus M})$ and the target distribution $p(y_i | v_i)$ as follows:

$$\sum_{j'=1}^S p(Y_{j'} | v_i) \log \frac{p(Y_{j'} | v_i)}{[f_{\text{VT}}(V_{\setminus M})]_{j'}} \quad (5.3)$$

We sum over all $i \in M$ and minimize with respect to θ . Following [93], we use only the top- k steps in $p(y_i | v_i)$ and set the probability of the remaining steps to 0.

Lin *et al.* [93] show that the distribution matching loss results in a slight improvement over step classification loss. For VideoTaskformer, we find both objectives to have similar performance and step classification outperforms distribution matching on some downstream tasks.

We use f_{VT} as a feature extractor (layer before softmax) to extract step representations for new video segments.

Downstream Tasks

To show that the step representations learned by VideoTaskformer capture task structure and semantics, we evaluate the representations on 6 downstream tasks—3 new tasks which we introduce (mistake step detection, mistake ordering detection, and long-term step forecasting) and 3 existing benchmarks (step classification, procedural activity recognition, and short-term step forecasting). We describe the dataset creation details for our 3 new benchmarks in Sec. 5.4.

Mistake Detection. A critical aspect of step representations that are successful at capturing the semantics and structure of a task is that, from these representations, *correctness* of task execution can be verified. We consider two axes of correctness: content (what steps are portrayed in the video) and ordering (how the steps are temporally ordered). We introduce 2 new benchmark tasks to test these aspects of correctness.

- **Mistake step detection.** The goal of this task is to identify which step in a video is incorrect. More specifically, each input consists of a video $V = \{v_1, \dots, v_K\}$ with K steps. V is identical to some unaltered video V_1 that demonstrates a correctly executed task, except that step v_j (for some randomly selected $j \in [1, \dots, K]$) is replaced with a random step from a different video V_2 . The goal of the task is to predict the index j of the incorrect step in the video.

- **Mistake ordering detection.** In this task, the goal is to verify if the steps in a video are in the correct temporal order. The input consists of a video $V = \{v_1, \dots, v_K\}$ with K steps. There is a 50% probability that V is identical to some (correctly ordered) video $V_1 = \{v_1^1, \dots, v_K^1\}$, and there is a 50% probability that the steps are randomly permuted. That is, $v_i = v_{\pi_i}^1$ for some random permutation π of indices $[1, \dots, K]$. The goal of the task is to predict whether the steps are ordered correctly or are permuted.

Step Forecasting. As another way to evaluate how learned step representations capture task structure, we test the capabilities of our model in anticipating future steps given one or more clips of a video.

- **Short-term forecasting.** Consider a video $V = \{v_1, \dots, v_n, v_{n+1}, \dots, v_K\}$ where v_i denotes a step, and V has step labels $\{y_1, \dots, y_K\}$, where $y_i \in Y$, the finite set of all step labels in the dataset. Short-term forecasting involves predicting the step label y_{n+1} given the previous n segments $\{v_1, \dots, v_n\}$ [93].

- **Long-term step forecasting.** We introduce the challenging task of long-term step forecasting. Given a single step v_i in a video $V = \{v_1, \dots, v_K\}$ with step labels $\{y_1, \dots, y_K\}$, the task is to predict the step labels for the next 5 steps, i.e. $\{y_{i+1}, y_{i+2}, \dots, y_{i+5}\}$. This task is particularly challenging since the network receives very little context—just a single step—and needs to leverage task information learned during training from watching multiple different ways of executing the same task.

Procedural Activity Recognition. The goal of this task is to recognize the procedural activity (i.e., task label) from a long instructional video. The input to the network is all the K video clips corresponding to the steps in a video, $V = \{v_1, \dots, v_K\}$. The task is to predict the video task label $t \in \mathcal{T}$ where \mathcal{T} is the set of all task labels for all the videos in the dataset.

Step Classification. In this task, the goal is to predict the step label $y_i \in Y$ given the video clip corresponding to step v_i from a video $V = \{v_1, \dots, v_K\}$. No context other than the single clip is given. Therefore, this task requires fine-grained recognition capability, which would benefit from representations that contain information about the context in which a step gets performed.

For all of the above tasks, we use the step and task label annotations as supervision. We show the “zero-shot” performance of VideoTaskformer by keeping the video model f_{vid} and the transformer layer f_{trans} fixed and only fine-tuning a linear head f_{head} on top of the output representations. Additionally, we also show fine-tuning results where we keep the base video model f_{vid} fixed and fine-tune the final transformer f_{trans} and the linear layer f_{head} on top of it. The network is fine-tuned using cross-entropy loss with supervision from the step labels for all downstream tasks.

Implementation Details

Step labels from Weak Supervision. To train VideoTaskformer, we require step annotations, i.e., step labels with start and end timestamps in the video, for a large corpus of instructional videos. Unfortunately, this is difficult to obtain manually and datasets that provide these annotations, like COIN and CrossTask, are small in size ($\sim 10\text{K}$ videos). To overcome this issue, the video speech transcript can be mapped to steps in WikiHow and used as a weak form of supervision [93]. The intuition behind this is that WikiHow steps are less noisy compared to transcribed speech.

The WikiHow dataset contains a diverse array of articles with step-by-step instructions for performing a range of tasks. Denote the steps across all T tasks in WikiHow as $s = \{s_1, \dots, s_N\}$, where s_n represents the natural language title of the n th step in s , and N is the number of steps across all tasks in WikiHow. Each step s_n contain a lengthy language-based description which we denote as y_n .

Consider a video with K sentences in the automatic speech transcript denoted as $\{a_1, \dots, a_K\}$. Each sentence is accompanied by a $\{start, end\}$ timestamp to localize it in the video. This yields K corresponding video segments denoted as $\{v_1, \dots, v_K\}$. Each video segment v_i is a sequence of F RGB frames having spatial resolution $H \times W$. To obtain the step label for a segment v_i , the

corresponding sentence in the transcript a_i is used to find the distribution of the nearest steps in the WikiHow repository. Following [93], we approximate this distribution using a textual similarity measure sim between y_n and a_i :

$$P(y_n|v_i) \approx \frac{\exp(\text{sim}(a_i, y_n))}{\sum_{n'} \exp(\text{sim}(a_i, y_{n'}))}. \quad (5.4)$$

The authors of [93] found it best to search over all the steps across all tasks (i.e., all y_n), rather than the set of steps for the specific task referenced in the video. The similarity function sim is formulated as a dot product between language embeddings obtained from a pre-trained language model.

Language model. To compare WikiHow steps to the transcribed speech via the sim function, we follow the same setup as in Lin *et al.* [93]. For a fair comparison to the baseline, we use MPNet (paraphrase-mpnet-base-v2) to extract sentence embeddings $\in \mathbb{R}^{768}$.

Video model. VideoTaskformer is a TimeSformer model with a two-layer transformer. Following [93], the TimeSformer is initialized with ViT [28] pre-trained on ImageNet-21K, and is trained on subsampled clips from HowTo100M (with 8 frames sampled uniformly from each 8-second span).

We include additional implementation details in the Supplemental.

5.4 Datasets and Evaluation Metrics

Pre-training. For pre-training, we use videos and transcripts from the HowTo100M (HT100M) [104] dataset and steps from the WikiHow dataset [9]. HT100M contains 136M video clips from 1.2M long narrated instructional videos, spanning 23k activities such as “gardening” and “personal care.” The WikiHow dataset contains 10,588 steps collected from 1059 WikiHow articles which are sourced from the original dataset [75].

Evaluation. All evaluation benchmarks use videos and step annotations from the COIN dataset [152]. COIN consists of 11,827 videos related to 180 different tasks and provides step labels with start and end timestamps for every video. We use a subset of 11,104 videos that were available to download.

As described in Sec. 5.3, we introduce 3 new benchmark tasks in this work: mistake step detection, mistake ordering detection, and long-term step forecasting.

Mistake Step Detection. For creating the mistake step detection dataset, for every video in the COIN dataset, we randomly replace one step with a step from a different video. The network predicts the index of the mistake step. We use the same train/validation/test splits as in COIN and report average accuracy of predicting the mistake step index on the test set.

Mistake Ordering Detection. We synthetically create the mistake ordering detection dataset by randomly shuffling the ordering of the steps in a given video, for 50% of the videos and train the network to predict whether the steps are in the right order or not. While creating the dataset, we repeatedly shuffle the input steps until the shuffled “mistake” order is different from the original valid order. Additionally, we compare the shuffled “mistake” order across all the videos in the same task, to ensure it doesn’t match any other video’s correct ordering of steps. Despite these two

pre-processing checks, there might be noise in the dataset. We report average prediction accuracy on the test split.

Long-term step forecasting. Given a video clip corresponding to a single step, long-term step forecasting involves predicting the step class label for the next 5 consecutive steps. If there are fewer than 5 next steps we append NULL tokens to the sequence. We compute classification accuracy as the number of correct predictions out of the total number of predictions, ignoring NULL steps. We again use the same splits in the COIN dataset.

Additionally, we evaluate on 3 existing benchmarks: **step classification** [152] - predicts the step class label from a single video clip containing one step, **procedural activity recognition** [152] - predicts the procedure/task label given all the steps in the input video, and **short-term step forecasting** [93] - predicts the class of the step in the next segment given as input the sequence of observed video segments up to that step (excluded).

5.5 Experiments

Model	Pre-training Supervision	Acc (%)
TSN (RGB+Flow) [151]	Supervised: action labels	36.5*
S3D [103]	Unsupervised: MIL-NCE on ASR	37.5*
ClipBERT [84]	Supervised: captions	30.8
VideoCLIP [169]	Unsupervised: NCE on ASR	39.4
SlowFast [35]	Supervised: action labels	32.9
TimeSformer [9]	Supervised: action labels	48.3
LwDS: TimeSformer [9]	Unsupervised: k -means on ASR	46.5
LwDS: TimeSformer	Distant supervision	54.1
VideoTF (SC)	Unsupervised: NN on ASR	47.0
VideoTF (DM)	Distant supervision	54.8
VideoTF (SC)	Distant supervision	56.5

Table 5.1: **Step classification.** We compare to the accuracy scores for all baselines. VideoTF (SC) pre-trained with step classification loss on distant supervision from WikiHow achieves state-of-the-art performance on the downstream step classification task. We report baseline results from [93]. * indicates results by fine-tuning on COIN

We evaluate VideoTaskformer (VideoTF) and compare it with existing baselines on 6 downstream tasks: step classification, procedural activity recognition, step forecasting, mistake step detection, mistake ordering detection, and long-term forecasting. Results are on the datasets described in Sec. 5.4.

Downstream Model	Base Model	Pre-training Supervision	Acc (%)
TSN (RGB+Flow) [151]	Inception [150]	Supervised: action labels	73.4*
Transformer	S3D [103]	Unsupervised: MIL-NCE on ASR	70.2*
Transformer	ClipBERT [84]	Supervised: captions	65.4
Transformer	VideoCLIP [169]	Unsupervised: NCE on ASR	72.5
Transformer	SlowFast [35]	Supervised: action labels	71.6
Transformer	TimeSformer [9]	Supervised: action labels	83.5
LwDS: Transformer	TimeSformer [9]	Unsupervised: k -means on ASR	85.3
LwDS: Transformer	TimeSformer	Distant supervision	88.9
LwDS: Transformer w/ KB Transfer	TimeSformer	Distant supervision	90.0
VideoTF (SC; fine-tuning) w/ KB Transfer	TimeSformer	Unsupervised: NN on ASR	81.2
VideoTF (SC; linear-probe) w/ KB Transfer	TimeSformer	Distant supervision	83.1
VideoTF (DM; linear-probe) w/ KB Transfer	TimeSformer	Distant supervision	85.7
VideoTF (SC) w/ KB Transfer	TimeSformer	Distant supervision	90.5
VideoTF (DM) w/ KB Transfer	TimeSformer	Distant supervision	91.0

Table 5.2: Accuracy of different methods on the **procedural activity recognition** dataset.

Downstream Model	Base Model	Pre-training Supervision	Acc (%)
Transformer	S3D [103]	Unsupervised: MIL-NCE on ASR	28.1
Transformer	SlowFast [35]	Supervised: action labels	25.6
Transformer	TimeSformer [9]	Supervised: action labels	34.7
LwDS: Transformer	TimeSformer [9]	Unsupervised: k -means on ASR	34.0
LwDS: Transformer w/ KB Transfer	TimeSformer	Distant supervision	39.4
VideoTF (SC; fine-tuned) w/ KB Transfer	TimeSformer	Unsupervised: NN on ASR	35.1
VideoTF (SC; linear-probe) w/ KB Transfer	TimeSformer	Distant supervision	39.2
VideoTF (DM; linear-probe) w/ KB Transfer	TimeSformer	Distant supervision	40.1
VideoTF (SC) w/ KB Transfer	TimeSformer	Distant supervision	41.5
VideoTF (DM) w/ KB Transfer	TimeSformer	Distant supervision	42.4

Table 5.3: Accuracy of different methods on the **short-term step forecasting** dataset.

Downstream Model	Base Model	Pre-training Supervision	Acc (%)
Transformer (ASR text) w/ Task label	MPNet		39.0
Transformer	SlowFast [35]	Supervised: action labels	15.2
Transformer	TimeSformer [9]	Supervised: action labels	17.0
Transformer w/ Task label	TimeSformer [9]	Supervised: action labels	40.1
LwDS: Transformer w/ Task label	TimeSformer	Distant supervision	41.3
VideoTF (DM)	TimeSformer	Distant supervision	40.2
VideoTF (DM) w/ Task label	TimeSformer	Distant supervision	46.4

Table 5.4: Accuracy of different methods on the **long-term step forecasting** dataset.

Downstream Model Order	Base Model	Pre-training Supervision	Mistake	Detection Step
Transformer (ASR text) w/ Task label	MPNet [142]		34.2	33.4
Transformer w/ Task Label	SlowFast [35]	Supervised: action labels	28.6	26.1
Transformer w/ Task label	TimeSformer [9]	Supervised: action labels	36.0	34.7
LwDS: Transformer	TimeSformer	Distant supervision	17.1	11.2
LwDS: Transformer w/ Task Label	TimeSformer	Distant supervision	37.6	31.8
VideoTF (SC)	TimeSformer	Distant supervision	20.1	15.4
VideoTF (DM) w/ Task label	TimeSformer	Distant supervision	40.8	34.0
VideoTF (SC; fine-tuned) w/ Task label	TimeSformer	Distant supervision	41.7	35.4

Table 5.5: Accuracy of different methods on the **mistake step and ordering detection** test dataset.

Baselines

We compare our method to state-of-the-art video representation learning models for action/step recognition. We fine-tune existing models in a similar fashion to ours on the 6 downstream tasks. We briefly describe the best-performing baseline, Learning with Distant Supervision (LwDS) [93].

- **TimeSformer (LwDS) [93]**. In this baseline model, the TimeSformer backbone is pre-trained on HowTo100M using the Distribution Matching loss (but without any masking of steps as in our model). Next, a single-layer transformer is fine-tuned on top of the pre-trained representations from the base model for each downstream task.

- **TimeSformer w/ KB transfer (LwDS) [93]**. For procedural activity recognition and step forecasting, the LwDS baseline is modified to include knowledge base transfer via retrieval of the most relevant facts from the knowledge base to assist the downstream task. We also include results by adding the same KB transfer component to our method, referenced as w/ KB Transfer.

- **Steps from clustering ASR text**. As an alternative to the weak supervision from WikiHow, we introduce an unsupervised baseline that relies only on the transcribed speech (ASR text) to obtain steps. [108] introduced an approach to segment a video into steps by clustering visual features along the time axis. It divides the video into non-overlapping segments and groups adjacent video segments together based on a similarity threshold. We adopt a similar approach but in the text space. We compute sentence embeddings for the ASR sentences and group adjacent sentences if their similarity exceeds the average similarity of all sentences across the entire video. We include ablations with different thresholds in the Supplemental.

Ablations

We evaluate our design choices by ablating different components of our model.

- **Base model**. We report results for different base video models for pre-training: S3D [103], SlowFast [35], TimeSformer [9] trained on HT100M, and TimeSformer trained on Kinetics. For short-term step forecasting, procedural activity recognition, and step classification, the results are from [93].

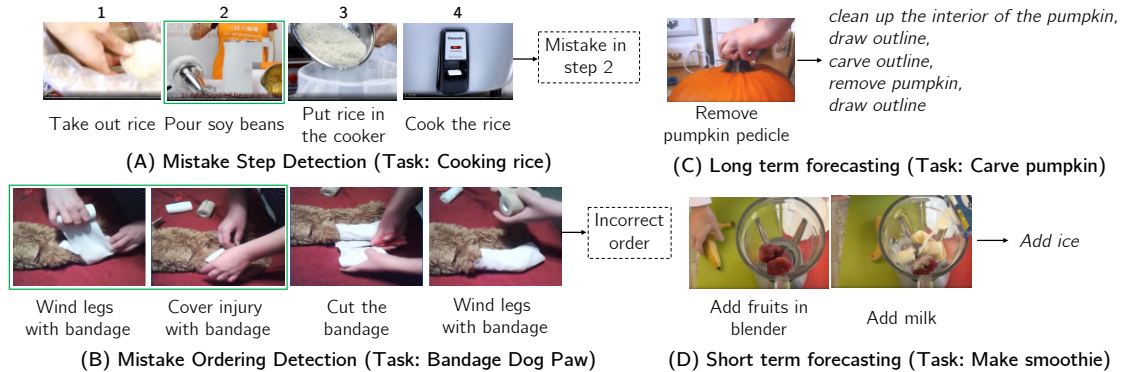


Figure 5.3: **Qualitative results.** We show qualitative results of our method on 4 tasks. The step labels are not used during training and are only shown here for illustrative purposes.



Figure 5.4: **Qualitative comparison.** We compare results from our method VideoTF to the baseline LwDS on the short-term forecasting task. Step labels are not passed to the model as input and are only for reference.

- **Loss function.** For pre-training VideoTF, we test both the loss functions, Step Classification (SC), and Distribution Matching (DM) described in Sec. 6.3.
- **Modalities.** For mistake step detection and long-term forecasting tasks, we tried replacing video features with ASR text during fine-tuning. The base model is a language model for embedding sentences in the ASR text and is kept fixed. The ASR text embeddings for all the segments of the video are fed as input to the downstream model, a basic single-layer transformer, which is fine-tuned to each of the tasks.
- **Task label.** For mistake detection and long-term forecasting tasks, we include the task name, e.g. “Install a Ceiling Fan”, as input to the downstream model. We compute the sentence embedding of the task label and append it to the list of video tokens fed as input to the model. This domain knowledge provides additional context which boosts the performance on these challenging downstream tasks.
- **Linear-probe vs Fine-tuning.** In linear-probe evaluation, only the f_{head} layer is fine-tuned to each downstream task and in the fine-tuning setting, all the layers of the segment transformer f_{trans} are fine-tuned.

Results

Quantitative Results. We compare our approach to several baselines on all downstream tasks. For all the downstream tasks, the downstream segment transformer is fine-tuned, except for linear-probe where we keep our pre-trained model fixed and only train a linear head on top of it for each downstream task.

On the step classification task in Tab. 5.1, VideoTF with step classification loss outperforms LwDS [93] by 2%, indicating that step representations learned with global context also transfer well to a task that only looks at local video clips. In procedural activity recognition (Tab. 5.2), we see that distribution matching loss works slightly better than step classification loss and our fine-tuned model achieves 1% improvement over the best baseline. For short-term forecasting in Tab. 5.3, we achieve a 3% improvement over LwDS and our unsupervised pre-training using NN with ASR outperforms previous unsupervised methods. We also note that linear-probe performance is competitive in Tab. 5.2 and outperforms baselines in Tab. 5.3. VideoTF with achieves a strong improvement of 5% over LwDS on the long-term forecasting task, 4% on mistake step detection, and 4% on mistake ordering detection. Adding task labels improves performance on all three tasks.

Additionally, we evaluate our approach on the activity recognition task in EPIC Kitchens-100 and include results in the Supplemental. We also report our models performance on the step localization task in COIN.

Qualitative Results. Fig. 5.3 shows qualitative results of our model VideoTF on the mistake detection tasks. Fig. 5.3 (A) shows a result on mistake step detection, where our model’s input is the sequence of video clips on the left and it correctly predicts the index of the mistake step “2” as the output. In (B), the order of the first two steps is swapped and our model classifies the sequence as incorrectly ordered. In (C), for the long-term forecasting task, the next 5 steps predicted by our model match the ground truth and in (D), for the short-term forecasting task, the model predicts the next step correctly given the past 2 steps. In Fig. E.2 we show an example result of our method compared to the baseline LwDS on the short-term forecasting task. Our method correctly predicts the next step as “remove air nozzle” since it has acquired knowledge of task structure whereas the baseline predicts the next step incorrectly as “install valve cap.”

5.6 Conclusion

In this work, we introduce a new video model, VideoTaskformer, for learning contextualized step representations through masked modeling of steps in instructional videos. We also introduce 3 new benchmarks: mistake step detection, mistake order detection, and long term forecasting. We demonstrate that VideoTaskformer improves performance on 6 downstream tasks, with particularly strong improvements in detecting mistakes in videos and long-term forecasting. Our method opens the possibility of learning to execute a variety of tasks by watching instructional videos; imagine learning to cook a complicated meal by watching a cooking show.

Acknowledgements. We would like to thank Suvir Mirchandani for his help with experiments and paper writing. This work was supported in part by DoD including DARPA’s LwLL, PTG and/or

CHAPTER 5. LEARNING AND VERIFICATION OF TASK STRUCTURE IN INSTRUCTIONAL VIDEOS

61

SemaFor programs, as well as BAIR's industrial alliance programs.

Chapter 6

Modular Visual Question Answering via Code Generation

6.1 Introduction

The scope of reasoning needed for visual question answering (VQA) is vast, and demands the synthesis of many skills – from grounding language to pixels [44, 123, 180] and spatial reasoning [61] to commonsense and knowledge-based reasoning [101]. Consider the question “*Is the carriage to the right of a horse?*”. To consistently answer such questions correctly, a system must recognize that the question is the conjunction of two subquestions: “*Is there a horse?*” and “*Is the carriage to the right of the horse?*” Scaling the typical finetuning paradigm to all possible combinations of reasoning skills is prohibitively expensive in annotation cost and makes it difficult to add skills to an already-trained system.

Modular approaches, on the other hand – from classic methods [77], to differentiable neural module networks (NMNs) [3, 59, 129]) – offer a potential route to leverage and scale to the compositional nature of visual reasoning as a means to generalize: *i.e. infinite use of finite means*. However, the modules of an NMN must still be trained jointly on a large dataset, and are also restricted in that they (i) require a parser, which must be modified if modules are added or removed from the system, and (ii) require retraining if a module is replaced.

In this work, we investigate an alternative class of modular VQA approaches, whereby building on the recent advent of highly capable out-of-the-box language models (LMs) [17, 114] and visual language models (VLMs) [87], we develop systems that formulate VQA as a program synthesis problem. Specifically, our method CodeVQA, illustrated in Figure 6.1, uses code-writing LMs to take questions as input, and outputs code to (i) orchestrate a series of visual primitive APIs that

This chapter is based on joint work with Sanjay Subramanian, Medhini Narasimhan, Kushal Khangaonkar, Kevin Yang, Arsha Nagrani, Cordelia Schmid, Andy Zeng, Trevor Darrell, and Dan Klein. It is presented much as it appeared in the: ACL 2023 proceedings.

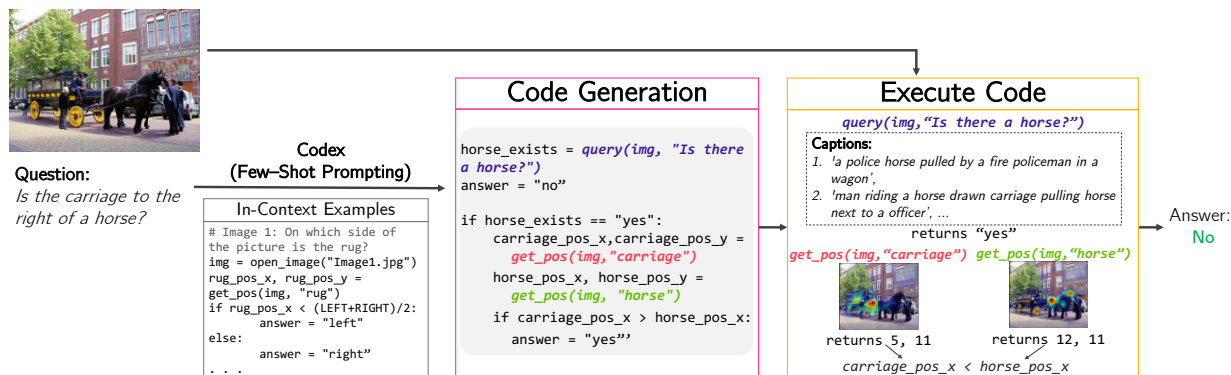


Figure 6.1: **CodeVQA Overview**. CodeVQA first prompts Codex with in-context examples that break down a given question into Python code. Using just the question, Codex generates an executable program that composes pre-defined visual modules using conditional logic, arithmetic, etc. The visual module, `query` answers a question by captioning the image and using an LM to answer based on the captions. `get_pos` retrieves the location of the object. Here, CodeVQA correctly identifies the question as a conjunction of a query and a spatial comparison and arrives at the right answer.

wrap around VLMs to probe the image for specific pieces of visual information (e.g. captions, pixel locations of entities, or image-text similarity scores), and (ii) reason about that information with the full expression of Python code (e.g. arithmetic, logic structures, feedback loops, etc.) to arrive at an answer. From a practical perspective, the modularity of CodeVQA combined with the few-shot prompting capabilities of LMs enable it to adapt to a broad range of desired VQA label distributions without additional model training, and benefits from replacing individual modules with improved versions as they become available.

We evaluate CodeVQA in the few-shot VQA setting, which has seen a great deal of recent work [1, 64, 172, 153]. Our method outperforms previous approaches by at least 3% on the COVR dataset [10], which requires reasoning over multiple images, and by 2% on the GQA dataset [61]. Our results suggest that the benefits of modularity with recent off-the-shelf models can be realized in VQA without additional model training.¹

6.2 Related Work

Several recent approaches for reasoning tasks consist of an LM that writes programs and an interpreter for these programs. Liang2022CodeAP applies this approach to robotics. binder introduces a framework for reasoning jointly over tables, text, and images, where the images are represented by image captions. subramanian-et-al-2022-reclip used a syntactic parser and hard-coded rules rather than an LM to aggregate outputs from CLIP [123] for zero-shot referring expression

¹Our code and annotated programs will be available at <https://github.com/sanjayss34/codevqa>.

comprehension; their finding that CLIP is not useful for spatial keywords motivates our code generation approach to spatial reasoning.

Concurrent with our work, other papers have introduced similar frameworks for multi-hop VQA [45, 148]. These papers conflate the benefit of program synthesis with the benefits of the LM, in-context examples, and vision models used as primitives. By contrast, we analyze the effect of program synthesis by comparing CodeVQA against a strong LM-based few-shot baseline using the same in-context example selection method. Moreover, while these frameworks rely on supervised VQA or object detection models, we show that we can obtain comparable performance (on the GQA dataset) using only the LM and models pre-trained on image-text pairs.

6.3 Few-shot VQA via Code Generation

In visual question answering (VQA), the inputs to the system are an image and a question and the output is a textual answer. We consider the few-shot VQA setting in which the system has access to only a small number (50) of human-annotated VQA instances.

Overview. Fig 6.1 illustrates our approach. Given an image and a corresponding question, CodeVQA first generates a Python program using just the question. It then executes this program, using the image when necessary, to predict the answer. We first define the set of code primitives that our system uses (§ 6.3). Then we describe how we generate a program that composes these primitives based on the question (§ 6.3). Finally, we enumerate the pre-trained models that we employ (§ 6.3).

Code Primitives

Primitives define basic operations over the image or over text that are often useful for VQA. In CodeVQA, we use three primitives, which are defined below. Each of these primitives is implemented using image-text matching (ITM), image-text contrastive (ITC), and image-captioning models, each of which can be trained with only image-caption pairs. The difference between ITM and ITC is that ITC computes separate image and text embeddings and takes a dot product, while ITM performs early fusion on the image and text features and is thus more computationally expensive. We note that our framework is not tied to this choice of primitives and can support other, more complex primitives that could draw on other aspects of the programming language and third-party libraries.

query(image, question) This function answers a question about the given image. Our implementation of this function is based on PnP-VQA [153] and PICa [172] and is implemented with the following steps: (1) using the ITM model, compute the GradCAM [133] between the question and the image (averaged over question tokens), (2) sample $K = 20$ image patches based on their GradCAM score, (3) generate a captions from the sampled patches using the captioning model, (4) Repeat steps (2) and (3) until C unique captions have been generated, and (5) predict the answer by prompting an LM with the question, captions, and in-context examples. The in-context examples in step (5) are selected as described in § 6.3. When the dataset involves reasoning over multiple images, each in-context example has the captions for all images.

get_pos(image, text) This function computes the GradCAM between the given text tokens and the image using the ITM model and returns the (x, y) pair that maximizes the GradCAM value. Note that this application of GradCAM is different from the one in query since we do not average over all question tokens. See Appendix E.1 for more information on how we compute GradCAM maps.

find_matching_image(images, text) In the setting where multiple images are associated with each question, there are questions that refer specifically to one image (e.g. “What is the woman holding?”). This function can be used to select the most relevant image from the set. It is implemented by scoring each image with the text using the ITC model and picking the image with the highest score.

Code generation

In the first stage of CodeVQA, we generate a Python program based on the question. Using Python over a domain-specific language is advantageous because (1) it supports arithmetic as well as control flow including loops and if statements [91]—all of which we use in our programs—and (2) large LMs for code generation (e.g. Codex [17]) have been trained on a large amount of Python code.

We construct a prompt that consists of an instruction, constants that define the dimensions of the image, and import statements and API documentation (as a code comment) that specify the available functions. In addition to the prompt, the input to the LM also includes expert-annotated programs for several in-context examples. An in-context example for few-shot prompting on the COVR dataset is shown below (question in gray, the program is highlighted).

```
# Image Set 1: How many images contain exactly 2
pink shoes??
images = open_images("ImageSet1.jpg")
count = 0
for an image in images:
    two_pink_shoes = query(image, "Are
        there exactly 2 pink shoes?")
    if two_pink_shoes == "yes":
        count += 1
answer = count
```

For an example of the rest of the prompt for the LM, see Appendix E. When executing the generated program results in a runtime error, we return call query on the image and the original question (including captions for all images if the instance involves multiple images).

Since all annotated programs cannot fit into a single input to the model, we must select which programs to use as in-context examples for each test question. Following [163], we use sentence embeddings² to retrieve the most similar questions for each test question.

²<https://huggingface.co/sentence-transformers/all-mpnet-base-v2>

Model	GQA Acc.	COVR Acc.	NLVR2 Acc.
Finetuned			
VisualBERT	–	57.9	67.0
VinVL-Base	65.1	–	83.1
Zero-shot			
FewVLM	29.3	–	–
PnP-VQA	42.3	–	–
BLIP-v2*	44.7	–	–
Few-shot			
FewVLM	35.7	–	–
VisProg*	50.5†	–	62.4
ViperGPT*	48.2	–	–
Few-shot PnP-VQA	46.6	45.8	63.4
CodeVQA (ours)	49.0	50.7	64.0

Table 6.1: **Results on GQA (testdev), COVR (test), and NLVR2 (test-public)** datasets from CodeVQA, Few-shot PnP-VQA, prior work VisualBERT [88], VinVL-Base [185], FewVLM [64], PnP-VQA [153], and concurrent work (marked with *) BLIP-v2 [86], VisProg [45], and ViperGPT [148]. Our method outperforms all few-shot methods from prior work. Highest few-shot scores for each full dataset are in **bold**. †indicates an evaluation on a stratified sample of the test set, which may not be directly comparable to results on the full test set.

Component models

Our approach relies on four pre-trained models: a code generation model, an ITM model, an ITC model, an IC model, and a question-answering LM for answering questions based on captions. We use the code-davinci-002 model [17] via the OpenAI API for both generating programs and for question-answering. We use the BLIP models [87] finetuned for ITM, ITC, and captioning.

6.4 Experiments

Implementation Details

See Appendix E.2 for implementation details.

Datasets

The GQA dataset [61] contains multi-hop questions generated from human-annotated scene graphs of individual images in Visual Genome [76]. The COVR dataset [10] contains multi-hop questions about *sets of images* in the Visual Genome and imSitu [173] datasets. These questions are synthetically generated from templates and are then paraphrased by humans. Unless otherwise specified, we

present results on the paraphrased questions. The NLVR2 dataset [146] contains statements about *pairs of images*, and the task is to determine whether each statement is true or false (we rephrase the statements as questions before feeding it to the methods that we evaluate). Appendix E.7 has further details about the datasets. For each of the three datasets, we wrote programs for 50 questions randomly sampled from the corresponding training set. Unless stated otherwise, we put 12 in-context examples in a prompt for a single-image dataset and 6 in-context examples in a prompt for a multi-image dataset (since including captions for multiple images increases the necessary context size for each example). We report the exact-match accuracies of the lower-cased answers.

Baseline

Our primary baseline is an adaptation of PnP-VQA [153] to the few-shot setting. We refer to it as “Few-shot PnP-VQA.” This baseline is equivalent to running the five-step query procedure described in § 6.3 for every question. We also compare to zero-shot and few-shot methods from prior and concurrent work. Appendix E.3 contains further details about those methods.

Results

Table 6.1 shows the results on the three datasets. CodeVQA has the highest accuracy among the few-shot techniques. Compared to Few-shot PnP-VQA, it has markedly better performance on COVR, which makes sense because in this dataset, the baseline approach must combine information across image captions for multiple images when given a single prompt. On the other hand, our method loops over the images and queries a single image at a time or selects the image most relevant to the question. Indeed, Table 6.3 shows that CodeVQA has the greatest advantage on instances involving 4 or 5 images. Compared to concurrent work that also uses program synthesis, CodeVQA generally performs better, which could be due to methodological differences like our in-context example retrieval or due to implementation details.

Fig. 6.2 shows a qualitative comparison of CodeVQA and the baseline Few-shot PnP-VQA on the COVR dataset. CodeVQA answers the question correctly by answering a simpler question for each image and comparing the answers, while Few-shot PnP-VQA answers incorrectly despite producing captions with the necessary information.

Ablations

Table 6.2 compares embedding-based retrieval of in-context examples with random retrieval. CodeVQA’s improvement over Few-shot PnP-VQA is greater when in-context examples are retrieved by embedding. Embedding-based retrieval offers a systematic way to collect relevant in-context examples rather than curating a single set of examples as in visprog.

In Appendix E.5, we include ablations for the question-answering LM and for the number of shots in the prompt as well as results on validation sets. Table E.1 shows that CodeVQA improves over Few-shot PnP-VQA when either `code-davinci-002` or `text-davinci-003` is used as the

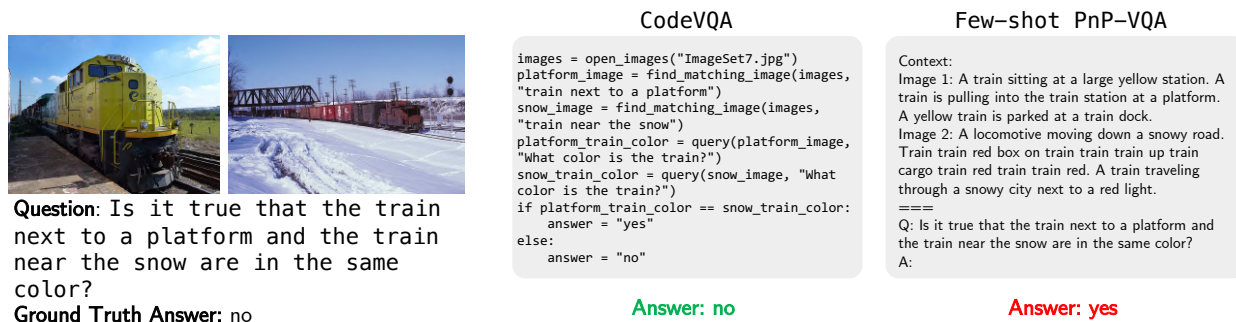


Figure 6.2: **Qualitative Comparison.** CodeVQA correctly answers this question from COVR by breaking it into simpler questions, answering each separately, and comparing the answers. Few-shot PnP-VQA answers incorrectly, even though the captions contain the necessary information. (Note that CodeVQA also generates captions, which are not shown here.)

question-answering LM. Table E.2 shows roughly constant accuracy as the number of in-context examples is varied.

Retrieval Method	Few-shot PnP-VQA	CodeVQA
<i>text-davinci-003</i>		
Random	48.15	49.9
Embedding	49.4	52.5
<i>code-davinci-002</i>		
Random	49.5	50.7
Embedding	52.1	55.3

Table 6.2: **Comparing Example Retrieval Techniques** on 2000 GQA validation examples. Italicized GPT model name denotes the model used as the question-answering LM.

Analysis

Figure 6.3 breaks down accuracy by question type. CodeVQA’s greatest improvement (roughly 30%) is in the subset consisting of questions about left/right or top/bottom object positions. There is also an improvement in “and” and “or” questions. This improvement could be related to the recent finding that LMs benefit from converting multi-hop into single-hop questions [120].³

We analyzed sources of error in CodeVQA on 100 examples in the COVR validation set for which CodeVQA answered incorrectly: irrelevant captions (31%), mistake in `find_matching_image` (12%), program generation error (14%), question-answering error (25%), predicted answer could

³Accuracy on this kind of question can also be improved by improving the LM. For instance, using *text-davinci-003* as the LM for QA closes the gap between Few-shot PnP-VQA and CodeVQA on “and” questions in GQA.

	Number of images				
	1	2	3	4	5
# of Instances	12	915	828	696	4440
Few-shot PnP-VQA	91.7	51.5	48.3	47.0	46.9
CodeVQA	75.0	53.3	48.7	53.2	53.4

Table 6.3: Accuracy by number of images per instance on COVR validation set.

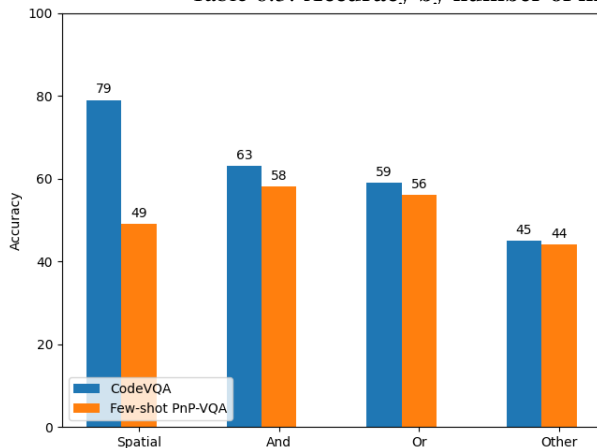


Figure 6.3: Accuracy by question type in GQA test set. CodeVQA (blue) outperforms Few-shot PnP-VQA (orange) on the spatial, and, or questions. “Spatial” refers to questions focusing on left/right or top/bottom relations or object positions.

be considered correct (14%), ground-truth is unclear/incorrect (16%), and numerical error (1%). Note that these categories are not mutually exclusive, and 13 of the 100 examples were marked with multiple categories. Thus, more errors are due to execution of the modules than program generation.

6.5 Conclusion

In this paper, we have introduced a framework for modular few-shot VQA. Our approach prompts an LM to generate a Python program that invokes pre-trained visual modules and composes the outputs of these modules to predict the answer. Unlike previous modular VQA techniques, this framework does not require (re-)training modules or a parser. Also, obtaining interpretable module outputs from previous modular approaches is nontrivial [145], whereas in our approach the modules are frozen and thus interpretable. CodeVQA can also be viewed as expanding pipelined systems [179] to the full expression of code. Our approach exhibits empirical gains, motivating future work on modular few-shot VQA.

6.6 Limitations

While the initial results are promising, the accuracy of our method remains lower than human VQA accuracy and models finetuned on the VQA datasets, which suggests that there may still be substantial progress that must be made before few-shot VQA methods with code synthesis are useful for practical real world applications. Also, further work is needed on extending the framework to additional primitives, as the results in Appendix E.6 show that doing so does not always lead to improvements over the baseline method. Another limitation of our approach is that it relies on large capable LMs, which may be restricted in use due to compute requirements or cost (e.g. via available APIs). We also focus in this work on benchmarking VQA capabilities with English as the primary language – future work may extend this to other languages via multilingual LMs.

6.7 Acknowledgements

We thank the members of the Berkeley NLP group, Grace Luo, and the anonymous reviewers for feedback on earlier drafts of this paper. We are grateful to Ben Bogin and Shivanshu Gupta for their assistance in evaluating CodeVQA and Few-shot PnP-VQA on the private COVR test set. SS, MN, and TD were supported in part by the DoD, including an NDSEG fellowship (for SS) and DARPA’s LwLL, PTG, and/or SemaFor programs, by the NSF, and/or by the Berkeley Artificial Intelligence Research (BAIR) industrial alliance program.

Chapter 7

Conclusion

In this thesis, I introduced multimodal models for understanding the semantics and structure of long videos. My work looked into synthesizing long video textures (20 - 30 minutes) and creating visual summaries of long videos (up to 2 hours). Next, I analyzed instructional videos in detail, looking into generating summaries in a self-supervised fashion and also detecting steps and mistakes in the videos. Finally, I explored the usage of large language models for visual question answering. Below I describe the key takeaways of each of these works followed by some thoughts on how the findings of this thesis can shape the future of long-term multimodal video understanding research.

In Chapter 2, we proposed a new learning-based technique, Contrastive Video Textures, which effectively captures video dynamics using an input-specific bi-gram model. This approach opens up possibilities for generating diverse and coherent textures of any length from short video clips. Unlike other generative methods, they don't suffer from resolution and generative artifacts and can generate infinite-length textures. We also showcased how video textures can be applied in conjunction with audio cues to create a more engaging and realistic visual experience.

Having synthesized long video textures from short clips, in Chapters 3 and 4 I focused on doing the reverse – creating short visual summaries of really long videos. With CLIP-It, we can now generate short summaries of any desired duration from videos as long as 4 hours. Our method also gives users the option to customize their summaries using natural language queries. However, video summarization benchmark datasets are quite small, as manually labeling the importance of every frame in a video is a tedious and cumbersome process. As a result, the ground-truth summaries are often subjective and noisy. In Chapter 4, we address these two issues with video summarization by centering on instructional videos. Using audio-visual alignment and similarity in task structure cues, we create a self-supervised dataset of 12,000 videos to train our Instructional Video Summarizer (IV-Sum) network. Additionally, we collect a clean test set by scraping *WikiHow videos* and no manual annotations. These datasets have since been used by the community as a standard for training and evaluating instructional video summarization [92, 12, 5, 6].

While summarizing instructional videos in Chapter 4, a prominent area that demanded further improvement emerged — learning meaningful step representations. Addressing this issue, Chapter 5 introduced TaskFormer, a novel masked-modeling transformer model designed for learning task-aware step representations in instructional videos. We empirically demonstrated that incorporating

the entire video to learn representations for individual steps resulted in improved performance for downstream tasks reliant on task-specific knowledge. Leveraging these learned representations, the model showcased the potential for identifying errors in the steps or their sequencing within new user-generated videos.

In Chapter 6, we propose CodeVQA, a novel framework for modular few-shot VQA that utilizes program synthesis to generate Python programs invoking pre-trained visual modules. Unlike previous approaches, our method does not require module (re-)training or a parser, ensuring interpretability. CodeVQA demonstrates significant empirical gains, showcasing the effectiveness of program synthesis for improving few-shot visual reasoning. The framework is easily scalable to various visual modules, and selecting relevant in-context examples is crucial for successful program synthesis. Our work motivates future research on advancing modular few-shot VQA techniques and extending the approach to other domains, pushing the boundaries of visual reasoning with program synthesis.

Given the success of large language models (LLMs) for decomposing questions into step-wise instructions, I have ventured into exploring their broader applications. My ongoing experiments reveal that LLMs can effectively transmute noisy transcripts from instructional videos into coherent recipes or step-by-step instructions without requiring manual annotations. This zero-shot approach relies solely on a prompt to instruct the LLM in generating steps from the transcript. We are actively investigating this capability for the automated generation of step-wise instructions and video summaries for the comprehensive HowTo100M dataset [104]. Furthermore, this novel approach has facilitated the training of video models using the extensive corpus of paired video-step data, a resource previously unavailable to the research community.

Below I list some challenges in long-term video understanding and interesting directions for future work.

Noisy transcript and alignment issues. One significant hurdle in video understanding arises from automatically generated transcripts, which often suffer from noise, and repetitive phrases. Demonstrators in these videos often tend to speak about the step before or after performing it, which results in a misalignment between the visual signals and the corresponding speech text. Additionally, inaccuracies in the generated timestamp labels further complicate the alignment between audio and visual content. To address these issues, recent works such as Temporal Alignment Networks [49] and LaViLA [189] propose techniques to enhance alignment by focusing on action-containing phrases or generating new sentences from the transcript. Exploring the potential of large language models (LLMs) to clean up transcripts holds promise as a future research direction, aiming to refine and improve the accuracy of input data for video understanding tasks.

How long is long-term? Designing video models for very long videos presents challenges related to memory and computational constraints, as processing extensive sequences demands significant resources and may cause memory overflow. Capturing long-range temporal dependencies and maintaining context in such videos is also difficult, as traditional models may struggle with complex temporal relationships. Additionally, representing hierarchical action structures becomes vital for understanding long videos effectively. Finally, retaining and leveraging long-term information is a challenge, as some critical details presented early in the sequence can be easily forgotten by models designed for shorter videos. Overcoming these challenges requires innovative architectural designs

and efficient data-handling techniques. My research addressed the challenge of processing very long videos by summarizing them into smaller segments. However, understanding all the intricate details of exceedingly long videos remains a significant task. Reevaluating the fixed frame rate used in current video processing may be necessary to develop models capable of analyzing such lengthy videos. Leveraging summarization techniques to extract key frames from relevant content regions can provide insights while managing computational complexities.

Alternate representations of long videos. Future research in video understanding can focus on exploring alternate representations for long videos, moving beyond conventional sequence networks [144], graph neural networks [157], regular transformers [4], and recently introduced hierarchical transformers [41]. The challenge lies in explicitly encoding the hierarchical layering of action information discussed in Chapter 1, encompassing atomic actions, steps, and events. Novel network architectures that effectively encode these hierarchical representations can significantly enhance video understanding. Additionally, considering task-specific abstraction levels is crucial; high-level representations suit video summarization and captioning tasks, mid-level representations benefit step representation learning or video question answering, while low-level representations are essential for actions and retrieval tasks. Tailoring representations to task requirements promises to improve the accuracy and efficiency of video analysis systems.

Bibliography

- [1] Jean-Baptiste Alayrac et al. “Flamingo: a Visual Language Model for Few-Shot Learning”. In: *ArXiv abs/2204.14198* (2022).
- [2] Jean-Baptiste Alayrac et al. “Unsupervised learning from narrated instruction videos”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.
- [3] Jacob Andreas et al. “Learning to Compose Neural Networks for Question Answering”. In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, June 2016, pp. 1545–1554. DOI: 10.18653/v1/N16-1181. URL: <https://aclanthology.org/N16-1181>.
- [4] Anurag Arnab et al. “Vivit: A video vision transformer”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2021.
- [5] Kumar Ashutosh et al. “Video-Mined Task Graphs for Keystep Recognition in Instructional Videos”. In: *arXiv preprint arXiv:2307.08763* (2023).
- [6] Kumar Ashutosh et al. “What You Say Is What You Show: Visual Narration Detection in Instructional Videos”. In: *arXiv preprint arXiv:2301.02307* (2023).
- [7] Siddhant Bansal, Chetan Arora, and CV Jawahar. “My View is the Best View: Procedure Learning from Egocentric Videos”. In: *European Conference on Computer Vision (ECCV)*. 2022.
- [8] Yizhak Ben-Shabat et al. “The ikea asm dataset: Understanding people assembling furniture through actions, objects and pose”. In: 2021.
- [9] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. “Is space-time attention all you need for video understanding?” In: *International Conference on Machine Learning (ICML)*. 2021.
- [10] Ben Bogin et al. “COVR: A Test-Bed for Visually Grounded Compositional Generalization with Real Images”. In: *Conference on Empirical Methods in Natural Language Processing*. 2021.
- [11] Piotr Bojanowski et al. “Weakly supervised action labeling in videos under ordering constraints”. In: *European Conference on Computer Vision (ECCV)*. 2014.

- [12] Andrea Burns et al. “A Suite of Generative Tasks for Multi-Level Multimodal Webpage Understanding”. In: *arXiv preprint arXiv:2305.03668* (2023).
- [13] Joao Carreira and Andrew Zisserman. “Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017).
- [14] Chien-Yi Chang et al. “D3TW: Discriminative differentiable dynamic time warping for weakly supervised action alignment and segmentation”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.
- [15] Chien-Yi Chang et al. “Procedure planning in instructional videos”. In: *European Conference on Computer Vision (ECCV)*. 2020.
- [16] Bor-Chun Chen, Yan-Ying Chen, and Francine Chen. “Video to Text Summary: Joint Video Summarization and Captioning with Recurrent Neural Networks.” In: *British Machine Vision Conference (BMVC)* (2017).
- [17] Mark Chen et al. “Evaluating Large Language Models Trained on Code”. In: *ArXiv abs/2107.03374* (2021).
- [18] Ting Chen et al. “A simple framework for contrastive learning of visual representations”. In: *arXiv preprint arXiv:2002.05709* (2020).
- [19] Xinlei Chen et al. “Improved baselines with momentum contrastive learning”. In: *arXiv preprint arXiv:2003.04297* (2020).
- [20] Yang Chen et al. “Mocycle-gan: Unpaired video-to-video translation”. In: 2019.
- [21] Aidan Clark, Jeff Donahue, and Karen Simonyan. “Efficient video generation on complex datasets”. In: *arXiv preprint arXiv:1907.06571* (2019).
- [22] Dima Damen et al. “Scaling egocentric vision: The EPIC-KITCHENS dataset”. In: *European Conference on Computer Vision (ECCV)*. 2018.
- [23] Abe Davis and Maneesh Agrawala. “Visual Rhythm and Beat.” In: *ACM Transactions on Graphics (TOG)* (2018).
- [24] Sandra Eliza Fontes De Avila et al. “VSUMM: A mechanism designed to produce static video summaries and a novel evaluation method”. In: *Patt. Rec. Letters* (2011).
- [25] Emily L Denton and Vighnesh Birodkar. “Unsupervised learning of disentangled representations from video”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.
- [26] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: 2019.
- [27] Li Ding and Chenliang Xu. “Weakly-supervised action segmentation with iterative soft boundary assignment”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

- [28] Alexey Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *International Conference on Learning Representations (ICLR)*. 2020.
- [29] Alexei A Efros and William T Freeman. “Image quilting for texture synthesis and transfer”. In: *ACM SIGGRAPH*. 2001.
- [30] Alexei A Efros and Thomas K Leung. “Texture synthesis by non-parametric sampling”. In: *IEEE International Conference on Computer Vision (ICCV)*. 1999.
- [31] Alexei A. Efros et al. “Recognizing Action at a Distance”. In: *IEEE International Conference on Computer Vision (ICCV)*. Nice, France, 2003, pp. 726–733.
- [32] Ariel Ephrat et al. “Looking to listen at the cocktail party: A speaker-independent audio-visual model for speech separation”. In: *ACM SIGGRAPH* (2018).
- [33] Jiri Fajtl et al. “Summarizing Videos with Attention”. In: *Asian Conference on Computer Vision (ACCV)* (2018).
- [34] Haoqi Fan et al. “Multiscale vision transformers”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2021.
- [35] Christoph Feichtenhofer et al. “Slowfast networks for video recognition”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2019.
- [36] Daniel Fried et al. “Learning to Segment Actions from Observation and Narration”. In: 2020.
- [37] Valentin Gabeur et al. “Multi-modal transformer for video retrieval”. In: *European Conference on Computer Vision (ECCV)* (2020).
- [38] Oran Gafni, Lior Wolf, and Yaniv Taigman. “Vid2Game: Controllable Characters Extracted from Real-World Videos”. In: *International Conference on Learning Representations (ICLR)*. 2020.
- [39] Leon Gatys, Alexander S Ecker, and Matthias Bethge. “Texture synthesis using convolutional neural networks”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2015.
- [40] Jort F Gemmeke et al. “Audio set: An ontology and human-labeled dataset for audio events”. In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2017.
- [41] Simon Ging et al. “Coot: Cooperative hierarchical transformer for video-text representation learning”. In: *NIPS* (2020).
- [42] Boqing Gong et al. “Diverse sequential subset selection for supervised video summarization.” In: *Advances in Neural Information Processing Systems (NeurIPS)* (2014).
- [43] Ian Goodfellow et al. “Generative adversarial networks”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2014.

- [44] Yash Goyal et al. “Making the v in vqa matter: Elevating the role of image understanding in visual question answering”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 6904–6913.
- [45] Tanmay Gupta and Aniruddha Kembhavi. “Visual Programming: Compositional visual reasoning without training”. In: *ArXiv abs/2211.11559* (2022).
- [46] Shir Gur, Sagie Benaim, and Lior Wolf. “Hierarchical Patch VAE-GAN: Generating Diverse Videos from a Single Sample”. In: *arXiv preprint arXiv:2006.12226* (2020).
- [47] Michael Gygli, Helmut Grabner, and Luc Van Gool. “Video summarization by learning submodular mixtures of objectives”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015).
- [48] Michael Gygli et al. “Creating summaries from user videos”. In: *European Conference on Computer Vision (ECCV)* (2014).
- [49] Tengda Han, Weidi Xie, and Andrew Zisserman. “Temporal alignment networks for long-term video”. In: *CVPR*. 2022.
- [50] Kaiming He et al. “Deep residual learning for image recognition”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).
- [51] Kaiming He et al. “Deep residual learning for image recognition”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.
- [52] Kaiming He et al. “Momentum contrast for unsupervised visual representation learning”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 9729–9738.
- [53] Xufeng He et al. “Unsupervised video summarization with attentive conditional generative adversarial networks”. In: *ACM international conference on Multimedia* (2019).
- [54] David J Heeger and James R Bergen. “Pyramid-based texture analysis/synthesis”. In: *ACM SIGGRAPH*. 1995, pp. 229–238.
- [55] Olivier J Hénaff et al. “Data-efficient image recognition with contrastive predictive coding”. In: *arXiv preprint arXiv:1905.09272* (2019).
- [56] Shawn Hershey et al. “CNN architectures for large-scale audio classification”. In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2017.
- [57] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* (1997).
- [58] Aleksander Holynski et al. “Animating Pictures with Eulerian Motion Fields”. In: *arXiv preprint arXiv:2011.15128* (2020).
- [59] Ronghang Hu et al. “Learning to reason: End-to-end module networks for visual question answering”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 804–813.

- [60] De-An Huang, Li Fei-Fei, and Juan Carlos Niebles. “Connectionist temporal modeling for weakly supervised action labeling”. In: *European Conference on Computer Vision (ECCV)*. 2016.
- [61] Drew A. Hudson and Christopher D. Manning. “GQA: A New Dataset for Real-World Visual Reasoning and Compositional Question Answering”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 6693–6702.
- [62] Vladimir Iashin and Esa Rahtu. “A better use of audio-visual cues: Dense video captioning with bi-modal transformer”. In: *British Machine Vision Conference (BMVC)* (2020).
- [63] Huaizu Jiang et al. “Super slomo: High quality estimation of multiple intermediate frames for video interpolation”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.
- [64] Woojeong Jin et al. “A Good Prompt Is Worth Millions of Parameters: Low-resource Prompt-based Learning for Vision-Language Models”. In: *ArXiv abs/2110.08484* (2021).
- [65] Nal Kalchbrenner et al. “Video pixel networks”. In: *International Conference on Machine Learning (ICML)* (2017).
- [66] Atsushi Kanehira et al. “Viewpoint-Aware video summarization”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018).
- [67] Hong-Wen Kang et al. “Space-time video montage”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2006).
- [68] Tero Karras, Samuli Laine, and Timo Aila. “A style-based generator architecture for generative adversarial networks”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.
- [69] Will Kay et al. “The kinetics human action video dataset”. In: *arXiv preprint arXiv:1705.06950* (2017).
- [70] Maurice George Kendall. “The treatment of ties in ranking problems”. In: *Biometrika* 33.3 (1945), pp. 239–251.
- [71] Changil Kim et al. “On learning associations of faces and voices”. In: *Asian Conference on Computer Vision*. 2018.
- [72] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *International Conference on Learning Representations (ICLR)* (2015).
- [73] Diederik P Kingma and Max Welling. “Auto-encoding variational Bayes”. In: *International Conference on Learning Representations (ICLR)*. 2013.
- [74] A Sophia Koepke et al. “Sight to Sound: An End-to-End Approach for Visual Piano Transcription”. In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020.
- [75] Mahnaz Koupaee and William Yang Wang. “Wikihow: A large scale text summarization dataset”. In: *arXiv:1810.09305* (2018).

- [76] Ranjay Krishna et al. “Visual Genome: Connecting Language and Vision Using Crowd-sourced Dense Image Annotations”. In: *International Journal of Computer Vision* 123 (2016), pp. 32–73.
- [77] Jayant Krishnamurthy and Thomas Kollar. “Jointly learning to parse and perceive: Connecting natural language to the physical world”. In: *Transactions of the Association for Computational Linguistics* 1 (2013), pp. 193–206.
- [78] Hilde Kuehne, Alexander Richard, and Juergen Gall. “Weakly supervised learning of actions from transcripts”. In: *CVIU*. 2017.
- [79] Anna Kukleva et al. “Unsupervised learning of action classes with continuous temporal embedding”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.
- [80] Vivek Kwatra et al. “Graphcut textures: image and video synthesis using graph cuts”. In: *ACM Transactions on Graphics (TOG)* (2003).
- [81] Frédéric Lavancier, Jesper Møller, and Ege Rubak. “Determinantal point process models and statistical inference”. In: *Journal of the Royal Statistical Society: Series B: Statistical Methodology* (2015).
- [82] Hsin-Ying Lee et al. “Dancing to music”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.
- [83] Yong Jae Lee, Joydeep Ghosh, and Kristen Grauman. “Discovering important people and objects for egocentric video summarization”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2012).
- [84] Jie Lei et al. “Less is more: ClipBERT for video-and-language learning via sparse sampling”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021.
- [85] Jie Lei et al. “MART: Memory-Augmented Recurrent Transformer for Coherent Video Paragraph Captioning”. In: *Association for Computational Linguistics* (2020).
- [86] Junnan Li et al. “BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models”. In: *ICML*. 2023.
- [87] Junnan Li et al. “BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation”. In: *ICML*. 2022.
- [88] Liunian Harold Li et al. “VisualBERT: A Simple and Performant Baseline for Vision and Language”. In: *ArXiv abs/1908.03557* (2019).
- [89] Yandong Li et al. “How local is the local diversity? reinforcing sequential determinantal point processes with dynamic ground sets for supervised video summarization”. In: *European Conference on Computer Vision (ECCV)* (2018).
- [90] Yitong Li et al. “Video Generation From Text.” In: *AAAI*. 2018.
- [91] J. Liang et al. “Code as Policies: Language Model Programs for Embodied Control”. In: *ArXiv abs/2209.07753* (2022).

- [92] Paul Pu Liang, Amir Zadeh, and Louis-Philippe Morency. “Foundations and recent trends in multimodal machine learning: Principles, challenges, and open questions”. In: *arXiv preprint arXiv:2209.03430* (2022).
- [93] Xudong Lin et al. “Learning To Recognize Procedural Activities with Distant Supervision”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022.
- [94] Tiecheng Liu and John R Kender. “Optimization algorithms for the selection of key frame sequences of variable length”. In: *European Conference on Computer Vision (ECCV)* (2002).
- [95] Ze Liu et al. “Video swin transformer”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022.
- [96] Ilya Loshchilov and Frank Hutter. “Decoupled Weight Decay Regularization”. In: *International Conference on Learning Representations (ICLR)*. 2018.
- [97] Zheng Lu and Kristen Grauman. “Story-driven summarization for egocentric video”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2013).
- [98] Yu-Fei Ma and Hong-Jiang Zhang. “A model of motion attention for video skimming”. In: *International Conference on Image Processing (ICIP)* (2002).
- [99] Behrooz Mahasseni, Michael Lam, and Sinisa Todorovic. “Unsupervised video summarization with adversarial LSTM networks”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017).
- [100] Arun Mallya et al. “World-Consistent Video-to-Video Synthesis”. In: *European Conference on Computer Vision (ECCV)*. 2020.
- [101] Kenneth Marino et al. “OK-VQA: A Visual Question Answering Benchmark Requiring External Knowledge”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 3190–3199.
- [102] Willi Menapace et al. “Playable Video Generation”. In: *arXiv preprint arXiv:2101.12195* (2021).
- [103] Antoine Miech et al. “End-to-end learning of visual representations from uncurated instructional videos”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.
- [104] Antoine Miech et al. “Howto100m: Learning a text-video embedding by watching hundred million narrated video clips”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2019.
- [105] Tomas Mikolov et al. “Distributed representations of words and phrases and their compositionality”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2013.
- [106] Ishan Misra, C Lawrence Zitnick, and Martial Hebert. “Shuffle and learn: unsupervised learning using temporal order verification”. In: *European Conference on Computer Vision (ECCV)*. Springer. 2016.

- [107] Medhini Narasimhan, Anna Rohrbach, and Trevor Darrell. “CLIP-It! language-guided video summarization”. In: *Advances in Neural Information Processing Systems (NeurIPS)* (2021).
- [108] Medhini Narasimhan et al. “TL; DW? Summarizing Instructional Videos with Task Relevance and Cross-Modal Saliency”. In: *European Conference on Computer Vision (ECCV)*. 2022.
- [109] Chong-Wah Ngo, Yu-Fei Ma, and Hong-Jiang Zhang. “Automatic video summarization by graph modeling”. In: *IEEE International Conference on Computer Vision (ICCV)* (2003).
- [110] Tae-Hyun Oh et al. “Speech2face: Learning the face behind a voice”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.
- [111] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. “Representation learning with contrastive predictive coding”. In: *arXiv preprint arXiv:1807.03748* (2018).
- [112] *Open video project*. <https://open-video.org/>.
- [113] Mayu Otani et al. “Rethinking the evaluation of video summaries”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2019).
- [114] Long Ouyang et al. “Training language models to follow instructions with human feedback”. In: *ArXiv abs/2203.02155* (2022).
- [115] Jungin Park et al. “SumGraph: Video Summarization via Recursive Graph Modeling”. In: *European Conference on Computer Vision (ECCV)* (2020).
- [116] Taesung Park et al. “Semantic Image Synthesis with Spatially-Adaptive Normalization”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.
- [117] Bryan A Plummer, Matthew Brown, and Svetlana Lazebnik. “Enhancing video summarization via vision-language embedding”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017).
- [118] Javier Portilla and Eero P Simoncelli. “A parametric texture model based on joint statistics of complex wavelet coefficients”. In: *International Journal of Computer Vision (IJCV)* (2000).
- [119] Danila Potapov et al. “Category-specific video summarization”. In: *European Conference on Computer Vision (ECCV)* (2014).
- [120] Ofir Press et al. “Measuring and Narrowing the Compositionality Gap in Language Models”. In: *ArXiv abs/2210.03350* (2022).
- [121] Yicheng Qian et al. “SVIP: Sequence VeriFication for Procedures in Videos”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022.
- [122] Zhaofan Qiu, Ting Yao, and Tao Mei. “Learning spatio-temporal representation with pseudo-3d residual networks”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2017.
- [123] Alec Radford et al. “Learning transferable visual models from natural language supervision”. In: *arXiv preprint arXiv:2103.00020* (2021).

- [124] Alexander Richard, Hilde Kuehne, and Juergen Gall. “Action sets: Weakly supervised action segmentation without ordering constraints”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.
- [125] Alexander Richard, Hilde Kuehne, and Juergen Gall. “Weakly supervised action learning with RNN based fine-to-coarse modeling”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.
- [126] Mrigank Rochan and Yang Wang. “Video summarization by learning from unpaired data”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2019).
- [127] Mrigank Rochan, Linwei Ye, and Yang Wang. “Video summarization using fully convolutional sequence networks”. In: *European Conference on Computer Vision (ECCV)* (2018).
- [128] Masaki Saito, Eiichi Matsumoto, and Shunta Saito. “Temporal generative adversarial nets with singular value clipping”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2017.
- [129] Raeid Saqr and Karthik Narasimhan. “Multimodal Graph Networks for Compositional Generalization in Visual Question Answering”. In: *Neural Information Processing Systems*. 2020.
- [130] Arno Schödl and Irfan A Essa. “Controlled animation of video sprites”. In: *ACM SIGGRAPH*. 2002.
- [131] Arno Schödl and Irfan A Essa. “Machine learning for video-based rendering”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2001.
- [132] Arno Schödl et al. “Video textures”. In: *ACM SIGGRAPH*. 2000.
- [133] Ramprasaath R. Selvaraju et al. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization”. In: *International Journal of Computer Vision* 128 (2016), pp. 336–359.
- [134] Fadime Sener and Angela Yao. “Unsupervised learning and segmentation of complex activities from video”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.
- [135] Fadime Sener et al. “Assembly101: A Large-Scale Multi-View Video Dataset for Understanding Procedural Activities”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022.
- [136] Ozan Sener et al. “Unsupervised semantic parsing of video collections”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2015.
- [137] Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. “Singan: Learning a generative model from a single natural image”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2019.
- [138] Aidean Sharghi, Boqing Gong, and Mubarak Shah. “Query-focused extractive video summarization”. In: *European Conference on Computer Vision (ECCV)* (2016).

- [139] Aidean Sharghi, Jacob S Laurel, and Boqing Gong. “Query-focused video summarization: Dataset, evaluation, and a memory network based approach”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017).
- [140] Michael A Smith and Takeo Kanade. “Video skimming and characterization through the combination of image and language understanding techniques”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (1997).
- [141] Michael A Smith and Takeo Kanade. *Video skimming for quick browsing based on audio and image characterization*. School of Computer Science, Carnegie Mellon University, 1995.
- [142] Kaitao Song et al. “MPNet: Masked and permuted pre-training for language understanding”. In: *Advances in Neural Information Processing Systems (NeurIPS)* (2020).
- [143] Yale Song et al. “TVSum: Summarizing web videos using titles”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015).
- [144] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. “Unsupervised learning of video representations using LSTMs”. In: *International Conference on Machine Learning (ICML)*. 2015.
- [145] Sanjay Subramanian et al. “Obtaining Faithful Interpretations from Compositional Neural Networks”. In: *Annual Meeting of the Association for Computational Linguistics*. 2020.
- [146] Alane Suhr et al. “A Corpus for Reasoning about Natural Language Grounded in Photographs”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 6418–6428. DOI: 10.18653/v1/P19-1644. URL: <https://aclanthology.org/P19-1644>.
- [147] Chen Sun et al. “VideoBERT: A joint model for video and language representation learning”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2019.
- [148] Dídac Surís, Sachit Menon, and Carl Vondrick. “ViperGPT: Visual Inference via Python Execution for Reasoning”. In: *arXiv preprint arXiv:2303.08128* (2023).
- [149] Christian Szegedy et al. “Going deeper with convolutions”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015).
- [150] Christian Szegedy et al. “Rethinking the inception architecture for computer vision”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.
- [151] Yansong Tang, Jiwen Lu, and Jie Zhou. “Comprehensive instructional video analysis: The COIN dataset and performance evaluation”. In: *IEEE transactions on pattern analysis and machine intelligence* (2020).
- [152] Yansong Tang et al. “Coin: A large-scale dataset for comprehensive instructional video analysis”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.
- [153] Anthony Meng Huat Tiong et al. “Plug-and-Play VQA: Zero-shot VQA by Conjoining Large Pretrained Models with Zero Training”. In: *Findings of ACL: EMNLP* (2022).

- [154] Sergey Tulyakov et al. “MoCoGAN: Decomposing motion and content for video generation”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.
- [155] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. “Deep image prior”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.
- [156] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in Neural Information Processing Systems (NeurIPS)* (2017).
- [157] Paul Vicol et al. “Moviegraphs: Towards understanding human-centric situations from videos”. In: *CVPR*. 2018.
- [158] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. “Generating videos with scene dynamics”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2016.
- [159] Qingyun Wang et al. “Multimedia Generative Script Learning for Task Planning”. In: *arXiv:2208.12306* (2022).
- [160] Ting-Chun Wang et al. “Few-shot video-to-video synthesis”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.
- [161] Ting-Chun Wang et al. “Video-to-Video Synthesis”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.
- [162] Ting-Chun Wang et al. “Video-to-video Synthesis”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.
- [163] Zhenhailong Wang et al. “Language Models with Image Descriptors are Strong Few-Shot Video-Language Learners”. In: *ArXiv abs/2205.10747* (2022).
- [164] Donglai Wei et al. “Learning and using the arrow of time”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 8052–8060.
- [165] Huawei Wei et al. “Video summarization via semantic attended networks”. In: *The Association for the Advancement of Artificial Intelligence Conference (AAAI)* (2018).
- [166] Li-Yi Wei and Marc Levoy. “Fast texture synthesis using tree-structured vector quantization”. In: *ACM SIGGRAPH*. 2000.
- [167] Li-Yi Wei et al. “State of the art in example-based texture synthesis”. In: 2009.
- [168] Saining Xie et al. “Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification”. In: *European Conference on Computer Vision (ECCV)*. 2018.
- [169] Hu Xu et al. “VideoCLIP: Contrastive Pre-training for Zero-shot Video-Text Understanding”. In: 2021.
- [170] Jingwei Xu et al. “Video Prediction via Example Guidance”. In: *International Conference on Machine Learning (ICML)* (2020).
- [171] Zhongwen Xu, Yi Yang, and Alex G Hauptmann. “A discriminative CNN video representation for event detection”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

- [172] Zhengyuan Yang et al. “An Empirical Study of GPT-3 for Few-Shot Knowledge-Based VQA”. In: *AAAI Conference on Artificial Intelligence*. 2021.
- [173] Mark Yatskar, Luke Zettlemoyer, and Ali Farhadi. “Situation Recognition: Visual Semantic Role Labeling for Image Understanding”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 5534–5542.
- [174] Yufei Ye et al. “Compositional video prediction”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2019.
- [175] Serena Yeung, Alireza Fathi, and Li Fei-Fei. “Videoset: Video summary evaluation through text”. In: *arXiv preprint arXiv:1406.5824* (2014).
- [176] Li Yuan et al. “Cycle-SUM: Cycle-consistent Adversarial LSTM Networks for Unsupervised Video Summarization”. In: *The Association for the Advancement of Artificial Intelligence Conference (AAAI)* (2019).
- [177] Rowan Zellers et al. “Merlot reserve: Neural script knowledge through vision and language and sound”. In: *CVPR*. 2022.
- [178] Rowan Zellers et al. “Merlot: Multimodal neural script knowledge models”. In: *NIPS* (2021).
- [179] Andy Zeng et al. “Socratic models: Composing zero-shot multimodal reasoning with language”. In: *arXiv preprint arXiv:2204.00598* (2022).
- [180] Xiaohua Zhai et al. “Lit: Zero-shot transfer with locked-image text tuning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 18123–18133.
- [181] Haotian Zhang et al. “Vid2player: Controllable video sprites that behave and appear like professional tennis players”. In: *arXiv preprint arXiv:2008.04524* (2020).
- [182] Ke Zhang, Kristen Grauman, and Fei Sha. “Retrospective encoders for video summarization”. In: *European Conference on Computer Vision (ECCV)* (2018).
- [183] Ke Zhang et al. “Summary transfer: Exemplar-based subset selection for video summarization”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).
- [184] Ke Zhang et al. “Video summarization with long short-term memory”. In: *European Conference on Computer Vision (ECCV)* (2016).
- [185] Pengchuan Zhang et al. “VinVL: Revisiting Visual Representations in Vision-Language Models”. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), pp. 5575–5584.
- [186] Bin Zhao, Xuelong Li, and Xiaoqiang Lu. “Hierarchical recurrent neural network for video summarization”. In: *ACM international conference on Multimedia* (2017).
- [187] Bin Zhao, Xuelong Li, and Xiaoqiang Lu. “Hsa-rnn: Hierarchical structure-adaptive rnn for video summarization”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

- [188] Bin Zhao and Eric P Xing. “Quasi real-time summarization for consumer videos”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2014).
- [189] Yue Zhao et al. “Learning Video Representations from Large Language Models”. In: *CVPR*. 2023.
- [190] Kaiyang Zhou, Yu Qiao, and Tao Xiang. “Deep reinforcement learning for unsupervised video summarization with diversity-representativeness reward”. In: *The Association for the Advancement of Artificial Intelligence Conference (AAAI)* (2018).
- [191] Luowei Zhou, Chenliang Xu, and Jason J Corso. “Towards automatic learning of procedures from web instructional videos”. In: *The Association for the Advancement of Artificial Intelligence Conference (AAAI)*. 2018.
- [192] Luowei Zhou et al. “End-to-end dense video captioning with masked transformer”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018).
- [193] Yipin Zhou et al. “Dance Dance Generation: Motion Transfer for Internet Videos”. In: *arXiv preprint arXiv:1904.00129* (2019).
- [194] Jun-Yan Zhu et al. “Unpaired image-to-image translation using cycle-consistent adversarial networks”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2017.
- [195] Dimitri Zhukov et al. “Cross-task weakly supervised learning from instructional videos”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.
- [196] Daniel Zwillinger and Stephen Kokoska. “CRC standard probability and statistics tables and formulae”. In: *CRC Press* (1999).

Appendix A

Chapter 2 Supplementary Material

This supplementary section is organized as follows:

1. Implementation details
2. Unconditional Video Texture Baselines
3. Video Quality Metric
4. Unconditional Contrastive Audio-Video Textures
5. Comparing Transition Probabilities

Additionally, we include the following videos in the Supp. folder (and here for easy viewing):

1. An overview video explaining our method, results, comparisons to baselines for both unconditional and conditional settings, and videos for Figure 1, Figure 4 and MocoGAN videos.
2. Videos for Figure 5 of the main paper.
3. `baseline_comparisons`: Additional results of comparisons to baselines for both conditional and unconditional settings.
4. `Interpolation`: Qualitative results of textures synthesized with and without interpolation.

A.1 Implementation Details

Video Encoding. We use the SlowFast feichtenhofer2019slowfast action recognition model pre-trained on Kinetics-400 kay2017kinetics for encoding the video segments. We divide the video into overlapping segments using a window of length 0.5 seconds and a stride of 0.2 seconds. Depending on the frame rate of the video, this yields segments with varying number of frames. Each of these segments is then encoded by SlowFast model into \mathbb{R}^{512} . Next the query and target are passed through two separate MLPs, each consisting of 3 linear layers interspersed with ReLU activations.

The MLP maintains the size of the embedding such that the final outputs, $\phi(S)$ and $\psi(S)$ are in \mathbb{R}^{512} . We initialize the SlowFast model with weights pretrained on Kinetics dataset and fine tune the whole network end-to-end. We use the SGD optimizer with a learning rate of $1e-4$.

We also experimented with alternate video encoding networks, such as I3D (RGB + Flow) and ResNet3D (RGB + Flow). Both these networks performed on par with SlowFast for our task and we decided to use SlowFast for the final model as it does not require optical flow to be computed and can thus be trained end-to-end on the raw data.

Audio Encoding. We embed the audio segments using the VGGish model `hershey2017cnn` pretrained on AudioSet `gemmeke2017audio`. We remove the last fully connected layer from the model and use the output of the final convolutional layer as audio features. The learned audio representations for the source audio segments $\varphi(A^c)$ and the conditioning audio segments $\varphi(A^e)$ are in \mathbb{R}^{128} .

Interpolation. We typically set the number of interpolated frames to be added to be 4. This increases the FPS of the synthesized video by a factor of 3 (i.e. 2 frames is converted to 4). When there is no jump, the frames are repeated 3 times, to ensure the overall FPS of the video is the same.

Temperature tuning and threshold. For training our Contrastive Video Texture model, we experimented with multiple values of temperature (τ) and found 0.1 to work the best. At test time, setting the temperature to 0.1 and threshold (t) to 100.0% regurgitates the original video, as the segment with the highest probability (i.e. the positive segment) is always chosen and all values below that are thresholded to 0. Increasing the temperature and decreasing the threshold increases the entropy and allows for more random transitions in the output. We found that the number of transitions vs the temperature is fairly constant across all videos and include details in Sec. A.5. Upon manually analyzing several synthesized textures, we found that temperatures in the range of 0.3 – 0.7 and thresholds in the range of 99.8 – 99.98 are optimal for synthesizing videos which are temporally smooth yet different from the original video. We synthesized video textures at different temperatures and observed that our model is capable of synthesizing multiple plausible output videos given a single input video.

In case of the Classic algorithm and its variants, we found that the temperature and threshold were not generalizable across videos and only a specific temperature (sigma) and threshold value resulted in temporally coherent textures. For the classic methods, the right hyperparameter values had to be chosen on a per video basis, by manually analyzing the synthesized textures over a range of hyperparameter values, which was quite tedious. Thus, “learning” feature representations and the distance metric as opposed to computing distances on raw pixels helped ease the task of choosing the right hyperparameters.

Combining T_a and T_v . Smaller values of α allow for better audio-video synchronization but at the cost of continuity in the video. For most results reported here, we set α to either 0.5 or 0.7.

A.2 Unconditional Video Textures: Baselines

We first provide an overview of the classic video texture algorithm introduced in [132] followed by the descriptions of the baselines.

- **Classic Video Textures:** The classic video textures algorithm proposed in [132] computes a distance matrix D of pairwise distances between all frames in the video. The distance is computed as the L2-norm of the difference in RGB values between pairs of frames. Next, the distance matrix D is filtered with a 2 or 4-tap filter with binomial weights to produce matrix D' . The stride used while filtering is 1. If the input video is short, oftentimes this approach would not be able to find good transitions from the last frame and reaches a dead end. To avoid this, they use Q-learning to predict the anticipated (increased) “future cost” of choosing a given transition, given the future transitions that such a move might necessitate. This gives rise to D'' . The transition probabilities P'' are computed from D'' as $P''_{i,j} = \exp(-D''_{i+1,j}/\sigma)$.

To synthesize a texture, a frame i is chosen at random. This is added to the output sequence of frames. After displaying frame i , the next frame j is selected according to $P_{i,j}$. To improve the quality of the textures and to suppress non-optimal transitions they adopt a two step pruning strategy. First, they choose the optimal transition with the maximum transition probability, next they set all probabilities below some threshold to zero and pick a random transition from the non-zero probabilities. The output sequence is generated one frame at a time.

We generate textures using the algorithm above. Following the convention in [132], we set sigma to be a small multiple of the average (non-zero) values in the distance matrix. We tune this small multiple and the threshold on the train set and use the same values on the test set.

For an apples-to-apples comparison, we fix some of the shortcomings of the classic algorithm and compare to these modified versions described below.

- **Classic+** During inference, the number of frames appended to the output texture is the stride with which the initial video was segmented. While this stride is 1 for the classic algorithm, it is greater than 1 for our contrastive method. To ensure the difference in perceptual quality isn’t due to just the changes in stride length, we modify the classic algorithm to increase the stride during inference to be the same as our contrastive model. The distance matrix is still computed pairwise between frames but instead of appending a single frame, we append stride number of frames to the output. This stride is set to be the same value as our contrastive model.
- **Classic++.** To further reduce the gap between classic+ and contrastive method, we apply a stride > 1 while filtering the distance matrix D with the tap filter. This is equivalent to the approach we use in contrastive, which is dividing the video into overlapping segments of window W and stride s .
- **Deep Classic:** Additionally, we also tried replacing the frame-wise features in the classic algorithm with learned representations from a pre-trained resnet.

A.3 Video Quality Metric

We report the average FVD uninterthiner2018towards computed between the original videos and the synthesized video textures for both the unconditional and conditional settings. As shown in Tab. A.1, our Contrastive method obtains the lowest FVD of and is the best performing method.

For the conditional setting, both Random Clip and Audio NN baselines involve extracting and replaying a portion of the original video, and hence receive the lowest FVD scores. Our method performs better than Classic+Audio and VRB.

Method	FVD ↓
Classic	379
Classic Deep	443
Classic+	208
Classic++	226
Contrastive	151

Table A.1: **Unconditional Video Texture Synthesis.** We report FVD scores for all the baselines and our method. A lower FVD score is better.

Method	FVD ↓
Classic+Audio	536
Random Clip	135
Audio NN	138
VRB	415
Contrastive	158

Table A.2: **Audio-Conditioned Video Textures.** We report FVD scores for all the baselines and our method. A lower score is better.

A.4 Unconditional Contrastive Audio-Video Textures

In this variant of our method, we combined audio features with the 3D video features to train our contrastive model. Assuming the input video has corresponding audio, we extract overlapping audio segments following the same approach used for video segments in Sec. 3 of the main paper. Next, we use two separate encoder heads for the query and target audio pairs. We then concatenate the audio encodings with the 3D video encodings and pass the fused features through a linear layer. The network is trained using contrastive learning, using Eq. 2 of the main paper. We observed that the resulting audio-video textures were smooth not only in the video domain but the audio domain as well. However, they were not as diverse as the ones trained with just video features. For this reason we used the video only model.

A.5 Comparing Transition Probabilities

We compare the **transition probability matrices** generated by both classic and contrastive methods. Fig. A.1 shows the transition probability matrices for two different videos generated by Classic (1a, 2a) and Contrastive (1b, 2b) methods. It can be observed from the diagonal lines in the figure

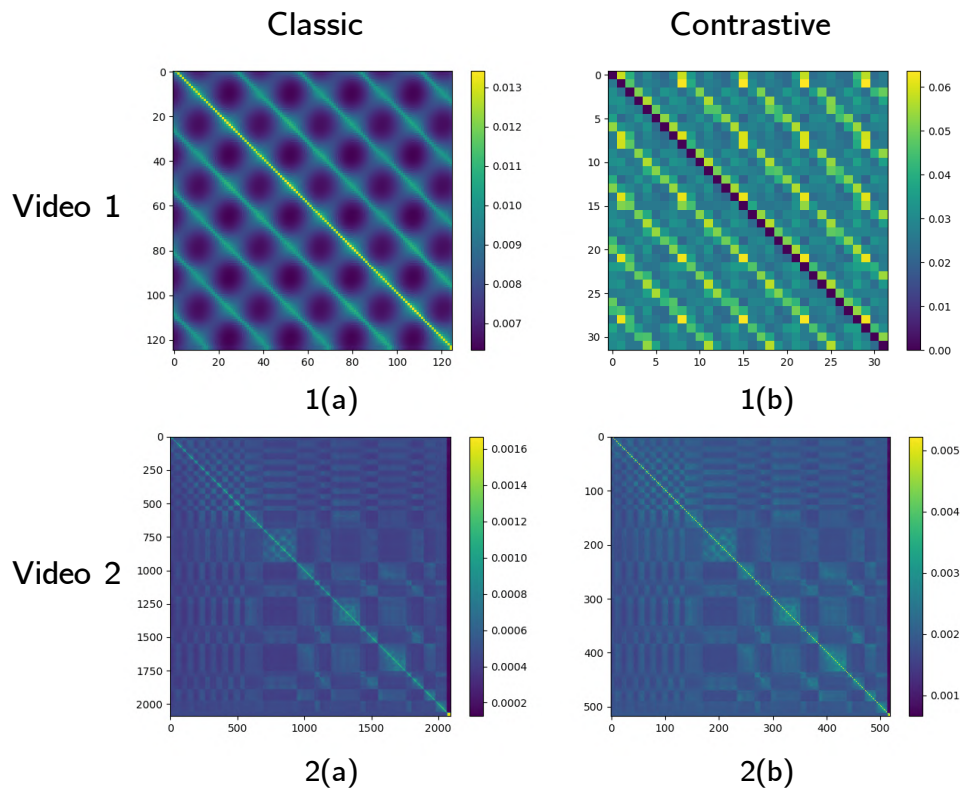


Figure A.1: Transition probability matrix for two different videos (in each row) for both classic and contrastive methods.

that the classic method assigns the same value to multiple frames whereas our method picks up on subtle differences and assigns different scores. This emphasizes that the distance metric learned by our method is better at distinguishing frames.

Fig. A.2 shows the variation in the number of transitions with sigma for the classic technique and with temperature for the contrastive technique. Number of transitions increases linearly with temperature for contrastive method whereas for the classic technique we found no such correlation. Moreover, a temperature of 0.3 and a threshold of 0.01 results in 15-20 jumps across all videos. There was no such strong correlation for the classic technique, making it necessary to tune hyperparameters on a per video basis.

Fig. A.3 shows some transitions in the video textures generated by contrastive model.

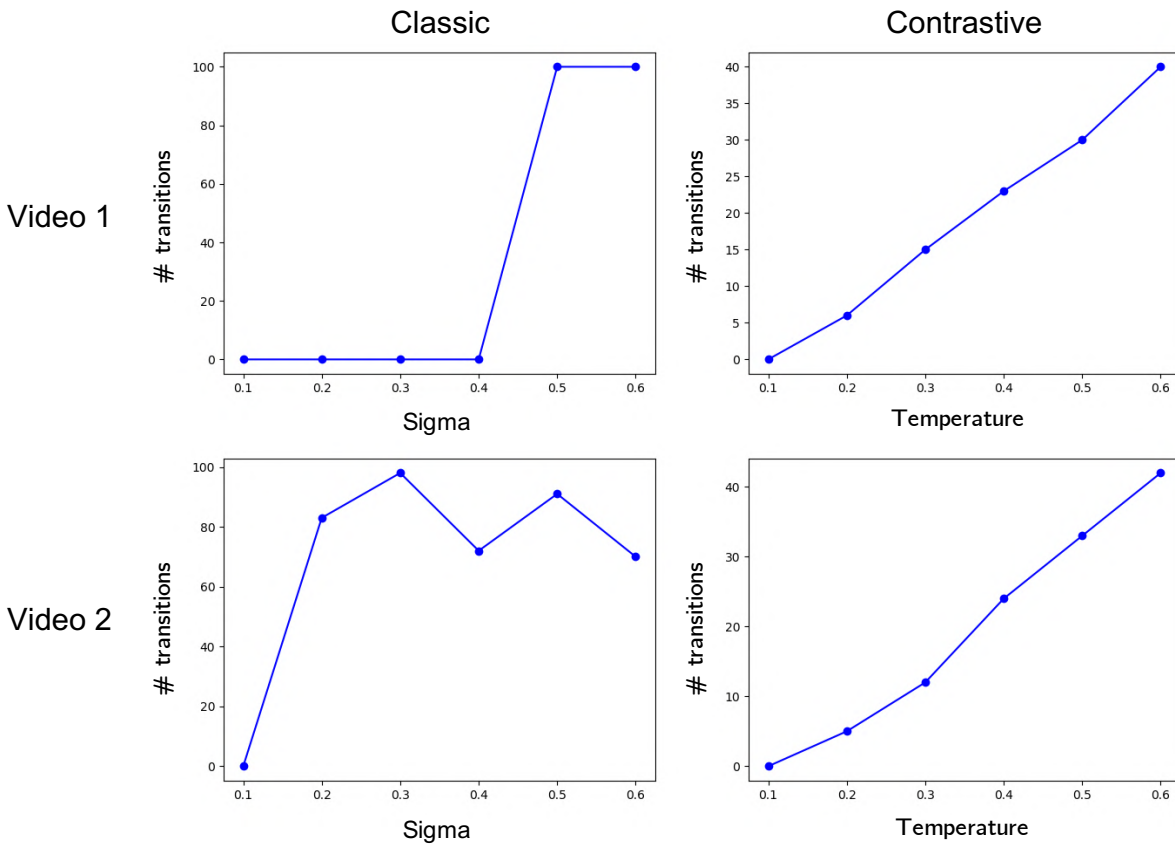


Figure A.2: Number of transitions vs Sigma for Classic and Number of transitions vs Temperature for Contrastive.

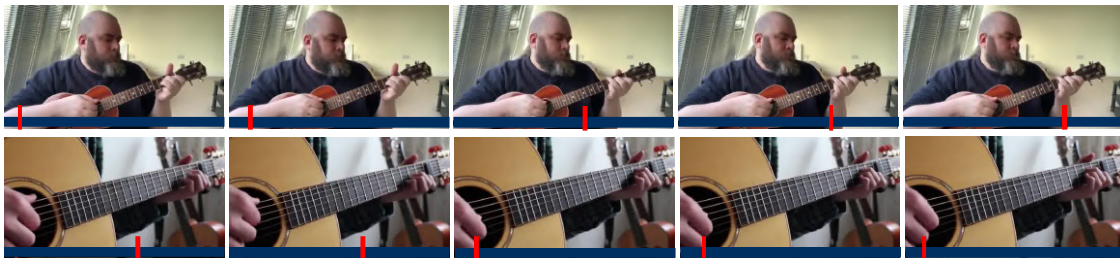


Figure A.3: The figure shows frames from two different videos synthesized by our method. Red bar indicates position of the original video being played. The transition happens at the third frame and is seamless in both cases. The first is a forward jump and the second is a backward jump.

Appendix B

Chapter 3 Supplementary Material

This section is organized as follows:

1. Implementation Details
2. Additional Results
3. Limitations

B.1 Implementation Details

Language-Guided Multi-head Attention. We use multi-head attention with 4 heads. We pass the Image Encoding as the Query and the Text Encoding as the Key and Value.

Frame-Scoring Transformer. We use a Transformer with 8 heads, 6 encoder layers and 6 decoder layers. The length of the sequence passed as input to the Transformer was heuristically chosen as 256.

Image Encoding. We encode the image using the CLIP [123] Image encoder to obtain image encoding $f_{img}(F) \in \mathbb{R}^{512}$.

Text Encoding. For query-focused video summaries, we encode the query using the CLIP Text encoder to obtain text embedding $f_{text}(C) \in \mathbb{R}^{512}$. For generic video summaries, we first generate dense video description using BMT [62] by sampling frames from the input video at 2 fps. For a 2-3 min video BMT generates 10-15 sentences. Next, we uniformly sample 7 sentences from the dense description corresponding to different video segments over time. Each sentence is then encoded using CLIP text encoder and the 7 embeddings are concatenated to obtain a feature vector. This is passed through a linear layer to obtain the input text embedding. Heuristically, we found that sampling 7 captions worked best for TVSum and SumMe datasets where the average duration of the videos is 2 mins. For generic summarization on the QFVS dataset (day long videos) reported above, the frames are extracted at 2 FPS and pass this through the BMT pipeline. This generates roughly 20 sentences and we then sampled 15 captions for each video since the videos are significantly longer.

Next, we uniformly sample M captions from the dense description corresponding to different video segments over time. Each caption is then encoded using CLIP Text encoder and the M embeddings are concatenated to obtain a feature vector in $R^{M \times 512}$. This is passed through a linear layer to obtain $f_{text}(C) \in \mathbb{R}^{512}$. We find that $M = 7$ works best.

Table B.1: Kendall’s τ [70] and Spearman’s ρ [196] correlation coefficients computed on the TVSum benchmark [143].

Method	Kendall’s τ	Spearman’s ρ
Zhang et al. [184]	0.042	0.055
Zhou et al. [190]	0.020	0.026
Park et al.(SumGraph) [115]	0.094	0.138
CLIP-It	0.108	0.147
Human	0.177	0.204

Training. Note that the caption generator, image and text encoders are kept fixed. The Language-Guided Multi-Headed Attention network and the Frame-Scoring Transformer are trained using Adam optimizer and a learning rate of 1e-4 and weight decay of 0.001.

Computational Resources. For each dataset and data setting, we train our method for 20 epochs with a batch size of 100 which takes about 2-3 hours on 5 NVIDIA RTX 2080 GPUs.

Frame Scores to Shot Scores. For Generic Video Summarization, different datasets provide ground-truth annotations in different formats. Following [183, 184], we obtain a single set of ground-truth keyframes (small subset of isolated frames) for each video. If a frame is selected to be a part of the summary, it is labeled 1, otherwise 0. The model is trained using keyframe annotations but evaluated on keyshots (interval-based subset of frames). For fair comparison, we follow [183, 184, 127] to convert the keyframes to keyshots.

For Query-Focused Video Summarization, the video is divided into shots of 5 seconds each [138]. Ground-truth annotations are available for each shot. While prior work predicts a single score per shot, we predict scores for each frame in the shot. In order to combine the scores for all frames in a shot we use two strategies: taking the max and taking the average. We found that taking the average of scores assigned to all frames in a shot to determine the shot score works best.

B.2 Additional Results

We also follow Otani *et al.* [113] and report results on rank based metrics, Kendall’s τ [70] and Spearman’s ρ [196] correlation coefficients in the Tab. B.1 for TVSum. They are computed by first ranking the frames in the video based on the predicted scores and the ground-truth scores and then comparing the two rankings. The correlation scores are computed by averaging over the individual results. We outperform all the baselines on these metrics as well.

Table B.2: F1 scores of **CLIP-It** for different loss ablations.

Method	SumMe			TVSum		
	Standard	Augment	Transfer	Standard	Augment	Transfer
\mathcal{L}_c	49.1	52.6	46.2	60.2	61.7	57.3
$\mathcal{L}_c + \mathcal{L}_r$	53.0	55.4	50.3	64.5	66.5	63.4
$\mathcal{L}_c + \mathcal{L}_d$	53.7	55.8	50.8	65.4	67.6	64.3
$\mathcal{L}_{unsup} = \mathcal{L}_d + \mathcal{L}_r$	52.5	54.7	50.0	63.0	65.7	62.8
$\mathcal{L}_{sup} = \mathcal{L}_c + \mathcal{L}_d + \mathcal{L}_r$	54.2	56.4	51.9	66.3	69.0	65.5

Table B.3: Ablating the Cross-Modal Attention module.

Method	SumMe			TVSum		
	Standard	Augment	Transfer	Standard	Augment	Transfer
CLIP-It (MLP)	50.6	51.08	48.1	63.0	65.8	61.4
CLIP-It (Cross-Modal Attn)	54.2	56.4	51.9	66.3	69.0	65.5

In Tab. B.2, we ablate the different loss functions described in Sec. 3 of the main paper and report results on TVSum and SumMe datasets. \mathcal{L}_c is the Classification loss, \mathcal{L}_r is the Reconstruction loss, and \mathcal{L}_d is the Diversity loss. Results shown are for the our full model, CLIP-It. Parameters α , β , and λ described in Sec. 3 are chosen heuristically and are set to 0.5, 0.3 and 0.2 respectively.

As we see, the Classification loss alone yields the lowest F1 scores. Adding the Reconstruction or Diversity losses improves performance. However, in the unsupervised setting, as the ground truth annotations cannot be used, we remove the Classification loss. This causes a slight drop in performance. Our method works best in the supervised setting, when all three losses are combined.

In Table B.3, we verify the effectiveness of the language-guided attention block by replacing this block with a basic MLP. As seen, the performance drops by 4% thus proving the need for the multi-headed attention between the two modalities.

B.3 Limitations

As described, we use large scale language models for video captioning [62] and feature extraction (CLIP [123]) which may have encoded some inappropriate biases that could propagate to our model. In particular, as CLIP was trained on 400M image-caption pairs sourced from the Web, we can not rule out the presence of biases or stereotypes which may propagate into how the video frames are scored within our method.

Appendix C

Chapter 4 Supplementary Material

This section is organised as follows:

1. *WikiHow Summaries* Data Collection
2. Implementation Details
3. Additional Results
 - a) Results on instructional videos in generic video summarization datasets
 - b) Step recall
 - c) Model architecture ablations
4. Additional Qualitative Results
 - a) Qualitative comparison of ground-truth, IV-Sum, CLIP-It, and Step AV
 - b) Pseudo summary vs IV-Sum summary
 - c) Pseudo summary vs step-localization annotations
 - d) Failure case

C.1 *WikiHow Summaries* Data Collection

We provide more details on the WikiHow Summaries data collection process. As described in Sec. 4.2 of the main paper, these are the main stages of the dataset creation: (1) Scraping WikiHow videos (2) Localizing images/clips in video (3) Ground-truth summary from localized clips (4) Manual verification. Fig. C.1 illustrates our data collection process. We show an example for the article “*Prepare Tofu*”. We localize each of the individual steps (images/clip) in the main video by comparing the ResNet features and obtain short localized clips. The clips are stitched together to form the summary. A handful of summaries with spurious lengths (too long or too short) are manually verified and corrected.

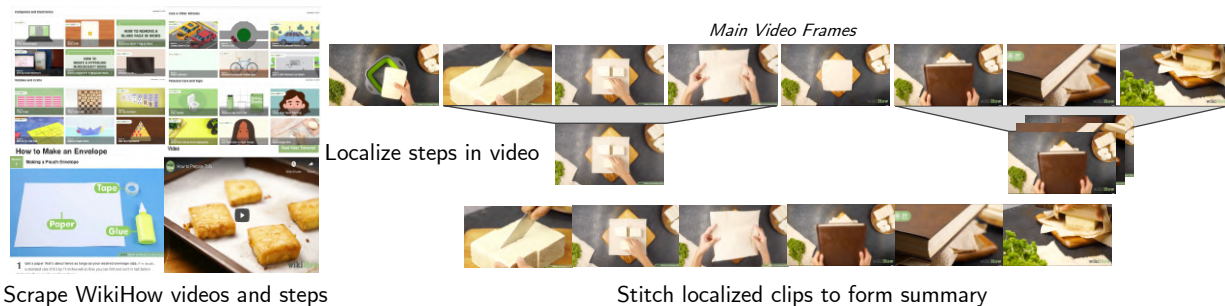


Figure C.1: **WikiHow Summaries Data Collection.** We first scrape all the main videos in the WikiHow articles, along with the images or video clips associated with each step. Next, the image/clip corresponding to each step is localized in the main video. The images are localized to ± 2.5 seconds (i.e. a 5 seconds window centered around the image). The localized clips are stitched together to form the summary.

We describe how we handle some edge cases in the articles, and the reasoning behind using ResNet features in stage (2).

Multiple methods. Sometimes the articles contain multiple methods of performing a task. If the video also contains multiple methods, as in this “*Draw a cow*” example, we localize each method in the video, and the summary is a compilation of all methods. The reasoning behind doing this is that users looking for a specific way of drawing a cow can take a quick glimpse of the summary and decide if they want to watch the whole video. However, if the article contains multiple methods but the video only contains one, as in *this* example, only the method depicted in the video is added in the summary.

Reason for using ResNet features instead of direct pixel comparison. As described in the main paper Sec 4.2, we compare ResNet features to localize the images/clips of the steps in the main video. The reason we compare ResNet features and not pixel values directly is because the images/clips associated with the steps aren’t always extracted from the main video. For example, in this article on “*Making a pinwheel*”, the frames in the images/clips are from a different video and don’t have exact matches in the *main video* for the article. Using ResNet features in place of pixels makes the localization robust to color/background changes, allowing us to localize steps despite an exact match of frames.

C.2 Implementation Details

Video processing. For generating pseudo summaries and for training IV-Sum, the videos are down-sampled to 8 FPS, and divided into non-overlapping segments of size 32 frames, which is the recommended segment size for MIL-NCE [103]¹. While training IV-Sum, we fix the number

¹We use the implementation of MIL-NCE available here https://github.com/antoine77340/MIL-NCE_HowTo100M

Table C.1: Hyperparameters for training IV-Sum.

Hyperparameter	Value
Batch size	24
Epochs	300
Learning rate IV-Sum	1e-3
Learning rate S3D fine-tuning	1e-4
Weight decay	1e-4
Dropout	0.1
Learning rate decay	StepLR
$t\%$	55%
#frames per segment	32
#frames per video during training	768
# Training FPS	8

of segments sampled from a video to be 28 (i.e. 896 frames) which are selected as a contiguous sequence from a randomly chosen start location. If the video is shorter in duration, it is padded with zeros. During inference, we retain the original fps of the video and all the segments are passed to IV-Sum. For concatenating the text representations to the visual representations, we follow the approach in MIL-NCE and map each visual clip to the sentences a few seconds before, after, and during the clip. The text embedding is an average of all the sentence embeddings.

Hyperparameters. Tab. C.1 shows detailed list of hyperparameters. For all baselines and our method, to ensure a fair comparison we generate the summary from scores by selecting the top $t\%$ of the highest scoring segments to be in the summary. t is set to be 55 based on the statistics in the validation set of WikiHow Summaries, where on average 55% of the original video appears in the summary.

Dimensions. We first describe the dimensions of each of the embeddings. The image embeddings are in $f_{\text{vid}}(s_i) \in \mathbb{R}^{512}$. The text embeddings for M transcript sentences using f_{text} are in $\mathbb{R}^{M \times 512}$ which are then fused using a 2 layer perceptron to \mathbb{R}^{512} . M is set to be the maximum number of sentences found in any ASR transcript. The image and text embeddings are concatenated and passed to the segment scoring transformer f_{trans} , the output dimension of this is in \mathbb{R}^{512} .

Computation resources. The training time is approximately 2 days using Distributed Data Parallel to train for 300 epochs on 8 NVIDIA RTX 2080 GPUs. The model inference time for a single video at its original fps is 1.5 minutes on average.

C.3 Additional Results

Evaluating on instructional videos in generic video summarization datasets. Here, we consider the existing generic video summarization datasets, in particular, those videos that fall under “in-

Table C.2: **Evaluating on generic video summarization datasets.** We compare F-Score of IV-Sum and CLIP-It on the instructional videos in TVSum.

Method	F-Score
CLIP-It [107]	0.72
IV-Sum	0.73

structional” domain, in order to validate our model further. Generic video summarization dataset TVSum [143] has 15 videos pertaining to the categories *changing a car tire*, *getting a car unstuck*, and *making a sandwich* while SumMe [48] has no instructional videos. We follow the evaluation protocol described in CLIP-It [107]. For a fair comparison to CLIP-It [107], we curate a test set by randomly selecting 7 of these 15 videos, while the remaining 8 are added to the training set, so as to ensure that the CLIP-It model sees instructional videos during training. The augmented training set is curated by combining the 8 videos with those in SumMe (25 videos), TVSum (45 videos), OVP [112] (50 videos), and YouTube [24] (39 videos). CLIP-It is trained on this augmented training set consisting of 168 videos (including 8 instructional videos) and evaluated on the held out 7 instructional videos. Our method is trained on pseudo summaries (built on top of CrossTask and COIN) and evaluated in a *zero-shot* way on the test set of 7 videos.

As seen in Tab. C.2, our method IV-Sum, although trained with noisy / weakly labeled pseudo summaries from a different data distribution, achieves an F-Score comparable to the CLIP-It [107], trained on human annotated summaries.

Table C.3: **Comparing step-recall.** We report step-recall on our method and 2 baselines.

Method	Step-recall
Step Cross-Modal Similarity	0.68
CLIP-It with ASR	0.70
IV-Sum	0.94

Step recall. We define an additional metric, *step-recall* to be the average percentage of steps present in the ground-truth summary which were successfully picked by the generated summary. Our *WikHow Summaries* dataset contains annotations of frames pertaining to each step, and if any of the frames from a step are present in the summary, we assume the step is covered. Using this logic, we generate a list of steps in the generated summary Y'_{step} , and a list of steps in the ground-truth Y_{step} . We compute step-recall as follows,

$$\text{Step-recall} = \frac{\text{overlap between } Y_{\text{step}} \text{ and } Y'_{\text{step}}}{\text{total duration of } Y_{\text{step}}}$$

Table C.4: **Instructional Video Summarizer Ablations.** We perform ablations on different components of the video summarizer network and report results on the *WikiHowTo Summaries* validation set.

(a) IVSum S3D backbone Ablations. We compare fixing the pre-trained weights of the S3D model to fine-tuning a part of it.		(b) IVSum Segment Scoring Transformer Ablations. We compare different architecture configurations of the segment scoring transformer.			
Method	F-Score	Method		F-Score	
S3D fixed	65.8	#heads	#layers		
S3D fine-tuned	67.9	SST	8	16	63.1
		SST	16	6	63.5
		SST	8	12	66.7
		SST	8	24	67.9
		MLP	-	-	32.1

Table C.5: **Loss Ablations.** We ablate different losses in the IV-Sum model and show results on validation set of *WikiHow Summaries*

Method	F-Score	Recall
MSE	67.9	84.5
MSE + Diversity	61.2	63.4
MSE + Reconstruction	67.6.	85.8

In Tab. C.3 we report the step-recall for Step Cross-Modal Similarity, CLIP-It with ASR (trained on generic video summarization datasets) and IV-Sum. Both Step Cross-Modal Similarity and CLIP-It with ASR baselines miss 30% of steps found in the ground truth summary while our method on average only misses 6% of the steps.

Loss Ablations. In Table C.5, we explore additional loss functions as in prior video summarization works [127, 126, 115, 107]. Diversity loss ensures diversity among the summary segments and the reconstruction loss enforces similarity in representations of the reconstructed summary and the input video. Adding diversity reduced the recall and we notice no improvement on adding reconstruction loss. We believe this may be because frames corresponding to different steps are not always diverse but are still important for the summary.

Model ablations. Table C.4a shows the performance comparisons between freezing the video and text encoding backbone (S3D) vs. fine-tuning part of the network. In Table C.4b, we ablate the segment scoring transformer (SST) in our model and change the number of encoder layers, heads, and also replace the transformer with an MLP. We report the F-Score on the validation set of *WikiHowTo Summaries*.

For a fair comparison of MIL-NCE vs CLIP features, we retrain our IV-Sum model replacing

video segments with frames and replacing MIL-NCE features with CLIP image and text features (same as the ones used in the CLIP-It baseline). We report results in Tab. C.6. We see that IV-Sum with CLIP performs at par with CLIP-It with ASR but falls short of IV-Sum, indicating the need to use video segments and MIL-NCE features pre-trained on HowTo100M.

Table C.6: **Instructional Video Summarization results on WikiHow Summaries.** All models were trained on pseudo summaries.

Method	F-Score		τ (Kendall)	ρ (Spearman)
	Val	Test	Test	Test
CLIP-It with ASR	62.5	61.8	0.093	0.191
IV-Sum with CLIP	61.8	62.0	0.094	0.201
IV-Sum	67.9	67.3	0.101	0.212

C.4 Additional Quantitative Results

Please watch the video on our website for qualitative results.

Comparison to baselines. We show video results comparing the ground-truth summary to that from IV-Sum (our method) and baselines Step Cross-Modal Similarity and CLIP-It with ASR trained on generic video summarization datasets. Our method picks all frames in the ground-truth and assigns high scores to salient frames. Step Cross-Modal Similarity misses the crucial step “*fold and tuck*” at the end as it assigns higher scores to irrelevant frames at the start of the video. This is because it has no knowledge of task-relevance. CLIP-It with ASR (trained on generic video summarization datasets) misses steps (like “*fold into a triangle*”) and assigns lower scores to the key frames in a step as it optimizes for diversity. **Evaluating Pseudo Summary generation procedure for WikiHow Summaries.** We found that there are 15 tasks which are shared between the Pseudo Summary training set and the WikiHow Summaries test set. We applied the method used to construct pseudo summaries to these 15 task videos in the WikiHow Summaries by fetching videos of the same task from our training set. We compare this to IV-Sum and report results in Tab C.7. We notice a slight improvement on all three metrics, indicating that our model is able to learn above the noise in the pseudo summaries.

Table C.7: **Evaluating pseudo summary generation on subset of WikiHow Summaries**

Method	F-Score	τ	ρ
	Test	Test	Test
Pseudo Summary Generation	38.0	0.03	0.36
IV-Sum	42.0	0.04	0.38

Pseudo summary vs IV-Sum summary. IV-Sum is trained on weakly labeled pseudo summaries that may sometimes be noisy. However, since the training loss is not 0, we check if our model learns despite the noise and produces summaries of a better quality. In this example, we show summaries for “*this*” video from the Pseudo Summaries dataset. As seen the pseudo summary contains an irrelevant segment where results from a web search are shown in Korean for nearly 10 seconds (9th second to the 19th second). The IV-Sum model trained on pseudo summaries yields a resulting summary without this segment, as it is able to learn “*task-relevance*” and “*cross-modal saliency*”.

Pseudo summary vs step-localization annotation. We compare pseudo summaries generated using our method to the step-localization summary. In the example “*Make a pumpkin spice latte*”, the input video can be found *here*. Step localization only localizes two main steps, “*boil milk*” and “*add coffee and blend*” whereas our pseudo summary contains all the main steps necessary to do the task.

Failure case. Since we always select the top 55% of the segments to be in the summary (i.e. $t=55\%$), the summary chosen by our method is sometimes much longer/shorter than the ground-truth summary. This is a failure case of the baseline methods as well. For example, for this 38 second video on “*How to ripen a cantaloupe*”, the ground-truth summary is a brief 15 seconds whereas our summary covers the steps in more detail and is 21 seconds long.

Appendix D

Chapter 5 Supplementary Material

In this section, we describe additional implementation details of our method and provide more qualitative results and comparisons on all the 6 downstream tasks.

D.1 Implementation Details

Pre-training. The base video model is a Timesformer [9] model with a ViT backbone initialized with ImageNet-21K ViT pretraining [28]. We pre-train our model on 64 A100 GPUs for 20 epochs which takes 120 hours for all the videos in the HowTo100M dataset. We use a batch size of 64 videos (1 video per GPU), each consisting of 12 segments. To train the model, we use SGD optimizer with momentum and weight decay. The learning rate is set to 0.01 and is decayed using a step learning rate policy of 10% decay at steps 15 and 19. We perform a second round of pretraining for 15 epochs using AdamW [96] with a learning rate of 0.00005.

We use a 15% masking ratio during pre-training. Segment transformer f_{trans} is a two layer transformer with 12 video segments as input. Each segment consists of 8 embedding vectors extracted from a series of 8 adjacent 8-second clips from the input video (spanning a total of 64 seconds). It has a 768 embedding dimension and 12 heads, along with learnable positional encodings at the beginning. The WikiHow knowledgebase has 10588 step classes all of which are used for training the network with step classification loss. For obtaining the distant supervision from WikiHow and mapping ASR text to step labels in the WikiHow knowledge base, we follow the setup described in [93].

Fine-tuning. For mistake step detection, mistake ordering detection, long term and short term step forecasting, and procedural activity recognition the input consists of 12 segments from the video. We fine-tune only the segment transformer f_{trans} and the linear head f_{head} using cross entropy loss, while the keeping the base TimeSformer video model f_{vid} as a fixed feature extractor. We use a learning rate of 0.005 with a step decay of 10% and train the network for 50 epochs using sgd optimizer.

For the step classification task, we only fine-tune the linear head, while keeping both the base video model and the 2 layer segment transformer fixed. We use a learning rate of 0.005 with a step

decay of 10% and train the network for 50 epochs using SGD optimizer.

D.2 Additional Quantitative Results

Activity Recognition. In Tab. D.2, we include results for activity recognition on EPIC-KITCHENS-100 by fine-tuning our pre-trained model for noun, verb, and action recognition tasks. We outperform all baselines on noun recognition, and are on par with MoViNet (Kondratyuk et al., CVPR 2021) on action recognition.

Model	Action (%)	Verb (%)	Noun (%)
MoViNet	47.7	72.2	57.3
LwDS: TimeSformer	44.4	67.1	58.1
VideoTaskformer (SC)	47.6	70.4	59.8

Table D.1: Activity Recognition on EPIC-KITCHENS-100.

Evaluating on step localization: We evaluate our pre-trained embeddings on the action segmentation task in COIN. Following previous work, we train a linear head on top of our fixed features and predict action labels for non-overlapping 1-second input video segments. LwDS attains 67.6% on this task, and our method achieves 69.1%.

Step labels as input: Our method uses visual features since step labels are not always available during inference. Nevertheless, for the purpose of comparison, we assume we have access to ground-truth step labels during inference and include results for all tasks. The results shown in Tab. D.2 are from training a single layer transformer on the COIN train set and evaluating on the test set, i.e. there is no pre-training. As expected, using step labels makes the task much simpler and it outperforms using visual features. However, adding task label information to visual features improves performance significantly for all the tasks.

Task	Step Labels(%)		Visual Features (%)	
	-	w/ Task label	-	w/ Task label
Short term forecasting	65	68	20	49
Long term forecasting	50	53	14	40
Mistake Ordering	80	82	60	65
Mistake Step	64	68	28	33

Table D.2: Step labels vs Visual features.

D.3 Additional Qualitative Results

Step Classification. We compare results from our method VideoTaskformer, to the baseline LwDS [93] in Fig. D.1. Since our model was trained on the entire video by masking out segments,

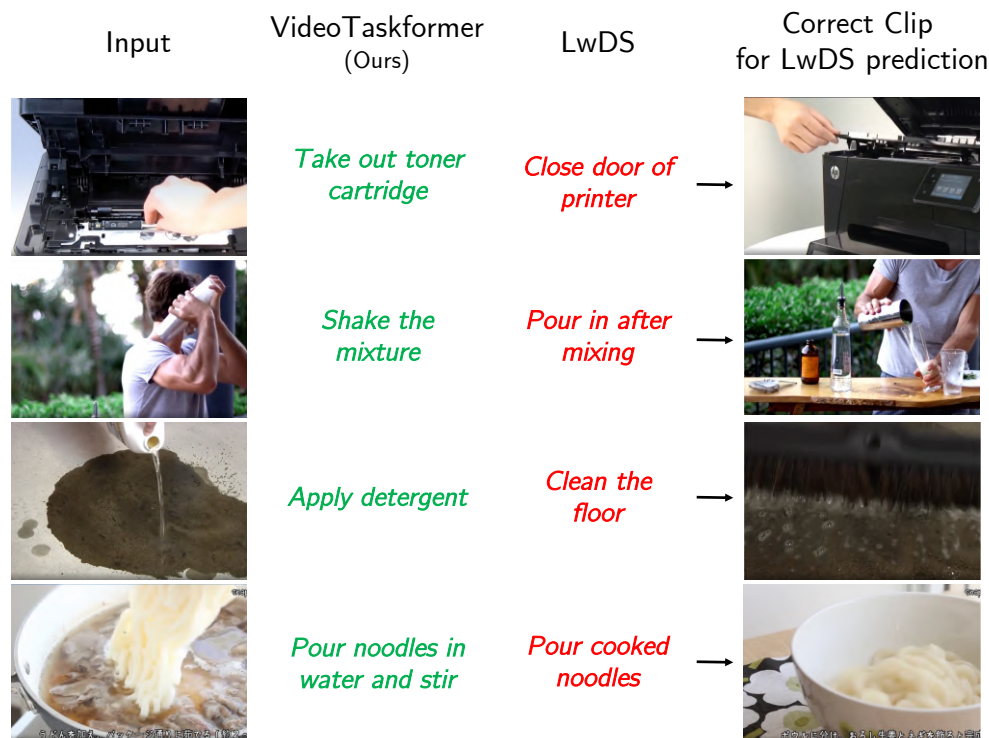


Figure D.1: **Step classification.** We qualitatively compare results from our method (VideoTaskformer) to the baseline LwDS on the step classification task. While the inputs are video clips, we only show a keyframe from the clip for visualization purposes. Correct predictions (VideoTaskformer) are shown in green and incorrect predictions (LwDS) are in red. We also show a frame from the clip corresponding to the incorrect prediction made by LwDS.

it has a better understanding of the relationship between different steps in the same task, i.e. learned representations are “*context-aware*”. As a result, it is better at distinguishing the steps within a task and correctly classifies all the steps in the four examples shown here. LwDS on the other hand incorrectly classifies all of the steps. For reference, we show a keyframe from the correct video step clip corresponding to the incorrect step class chosen by LwDS. The input image clips and the correct clips for the LwDS predictions are closely related and contain similar objects of interest, they correspond to different stages of the task and contain different actions. Since our model learns step representations “*globally*” from the whole video, it is able to capture these subtle differences. **Mistake Ordering Detection.** Fig. D.2 compares results of our method VideoTaskformer to the baseline LwDS on the mistake ordering detection task. We show two examples, “*lubricate a lock*” and “*change guitar string*”, where the steps in the input are swapped as shown by red arrows. Our method correctly detects that the input steps are in the incorrect order whereas the baseline predicts the ordering to be correct. As seen, detecting the order requires a high level understanding of the task structure, which our model learns through masking.







Input			Ground Truth	LwDS	VideoTaskformer (ours)
			<i>Incorrect order</i>	<i>Correct order</i>	<i>Incorrect order</i>
Task: Lubricate A Lock Apply lubricant → Insert key repeatedly → Wipe off excessive lubricant					
			<i>Incorrect order</i>	<i>Correct order</i>	<i>Incorrect order</i>
Task: Change Guitar Strings Fix the new string on the head of the guitar → Fix the new string on the lower part of the guitar → Adjust tightness of the string					

Figure D.2: **Mistake Order Detection.** Qualitative comparison of results from VideoTaskformer to LwDS. Step and task labels shown along with the input are for visualization purpose only. Correct answers are shown in green and incorrect answers in red.





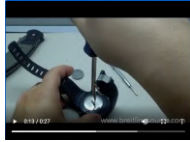
Input				
Step labels (for visualization only)	<i>Check type of back cover</i>	<i>Insert paper clip In hole</i>	<i>Replace battery</i>	<i>Install back cover</i>
Step Indices	0	1	2	3
Ground Truth	1			
LwDS	3	<i>Open the back cover</i>		
VideoTaskformer (ours)	1	Correct step for visualization		

Figure D.3: **Mistake Step Detection.** Qualitative comparison of results from VideoTaskformer to LwDS. Step and task labels shown along with the input are for visualization purpose only. Correct answers are shown in green and incorrect answers in red.


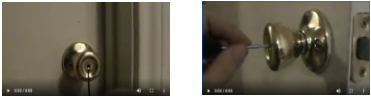

Task	Input	Ground Truth	LwDS	VideoTaskformer (ours)
Procedural Activity Recognition	 <p>Clean window surface Take off front of sticker Put on sticker Press sticker Tear off other side of sticker</p> <p>Task label: Paste car sticker</p>	Paste car sticker	Remove scratches from windshield	Paste car sticker
Short-Term Step Forecasting	 <p>1. Insert paper clip into lock 2. Twist paper clip by hand</p> <p>Task label: Open lock with paper clips</p>	3. Insert paper clip into lock	3. Install the new doorknob	3. Insert paper clip into lock
Long-Term Step Forecasting	 <p>1. Unscrew the screws used to fix the screen</p> <p>Task label: Replace laptop screen</p>	2. Pull out screen connector, 3. Remove the screen, 4. Install new screen, 5. Reset and screw on screw	2. Unscrew the screws, 3. Reset and screw on screw	2. Pull out screen connector, 3. Remove the screen, 4. Install new screen, 5. Reset and screw on screw

Figure D.4: Qualitative results for **procedural activity recognition, short term step forecasting, and long term step forecasting**. Step and task labels shown along with the input are for visualization purpose only. Correct answers are shown in green and incorrect answers in red.

Mistake Step Detection. Qualitative comparison on the mistake step detection task is shown in Fig. D.3. The input consists of video clip steps for the task “change battery of watch”. The second step is swapped with an incorrect step from a different task. Our method correctly identifies the index of the mistake step 1, whereas the baseline predicts 3 which is incorrect. We show the correct step for visualization purposes.

Procedural Activity Recognition. A result is shown in Fig. E.3. VideoTaskformer’s representations are context-aware and can identify the right task given the sequence of clips, “paste car sticker”. The baseline misidentifies the task as an incorrect similar task, “remove scratches from windshield”.

Short-term Step Forecasting. Fig. E.3 shows an input consisting of two clips corresponding to the first two steps for the task “open lock with paper clips”. The clips are far apart temporally, so the model needs to understand broader context of the task to predict what the next step is. Our method VideoTaskformer correctly identifies the next step as “insert paper clip into lock” whereas the baseline incorrectly predicts a step “install the new doorknob” from another task.

Long-term Step Forecasting. In Fig. E.3 we compare the future steps predicted by our model and the baseline LwDS on the long-term step forecasting task. Both models only receive a single clip as input, corresponding to the first step “unscrew the screws used to fix the screen” of the task “replace laptop screen”. Our model predicts all the next 4 ground-truth steps correctly, and in the right order. The baseline on the other hand predicts steps from the same task but in the incorrect order.

All of the above qualitative results further support the effectiveness of learning step representations through masking, and show that our learned step representations are “context-aware” and possess “global” knowledge of task-structure.

Appendix E

Chapter 6 Supplementary Material

E.1 GradCAM

Our computation of GradCAM follows prior work that uses vision transformers pnpvqa,albef. We are given a question with tokens q_1, \dots, q_T and an image that is tokenized into $K \times K$ patches. We use layer $L = 6$ to compute GradCAM, following pnpvqa. We compute a GradCAM map for each token as follows. Let $C \in \mathbb{R}^{T \times K^2}$ be the cross-attention map from layer L . Let $G \in \mathbb{R}^{T \times K^2}$ be the gradient of the image-text matching score with respect to C . Then the GradCAM map for token i is given by the i th row of $C \odot \text{ReLU}(G)$, where \odot denotes elementwise multiplication. As stated in Section 6.3, for the query primitive, we take the average GradCAM map across all question tokens, whereas for the get_pos primitive, we take the average GradCAM map across the input text tokens (which are part of the question tokens).

E.2 Implementation Details

To generate captions for in-context examples in each dataset, we run steps 1 – 4 for each of the 50 questions in the database of in-context examples. For GQA experiments, we use $C = 7$ captions per image, and for COVR experiments, where each question is associated with multiple images, we use $C = 3$ captions per image.¹ We use $C = 7$ captions for the NLVR2 dataset. Each reported accuracy result represents a single evaluation run over the corresponding evaluation set. For NLVR2 and some instances of COVR, the text input is a statement (to be classified as True/False). We convert each such statement to a question by adding the prefix “Is it true that” to it and converting the answer to “yes”/“no.” We use question embeddings to select 12 examples for GQA and 6 examples for COVR and NLVR2.

¹We chose this number of captions to be the maximum possible subject to the number of shots and the context size of the davinci model, which we used as our question-answering LM in preliminary experiments.

Model	GQA			COVR			
	Shots	Val Sample	Testdev	Shots	Val Sample	Val	Test
Few-shot PnP-VQA w/ text-davinci-003	12	49.4	44.9	6	51.4	–	–
CodeVQA (ours) w/ text-davinci-003	12	52.5	46.8	6	54.4	–	–
Few-shot PnP-VQA w/ code-davinci-002	12	52.1	46.6	6	49.0	47.8	45.8
CodeVQA (ours) w/ code-davinci-002	12	55.3	49.0	6	54.5	52.9	50.7

Table E.1: **Validation and test results on GQA and COVR.** OpenAI model name (text-davinci-003 or code-davinci-002) denotes which model was used as the question-answering model. GQA validation sample contains 2000 examples from the GQA validation set. COVR validation sample contains 1000 examples from the COVR non-paraphrased validation set. Highest scores on are in **bold**.

E.3 Details on Baselines

FewVLM randomly samples 16 few-shot examples for GQA. VisProg runs the program generation and execution pipeline five times, and for each iteration randomly samples 24 few-shot examples for GQA and 16 few-shot examples for NLVR2. ViperGPT uses 8 few-shot examples for GQA. VisProg uses text-davinci-002 or text-davinci-003 for code generation (according to the code release), while ViperGPT uses code-davinci-002.

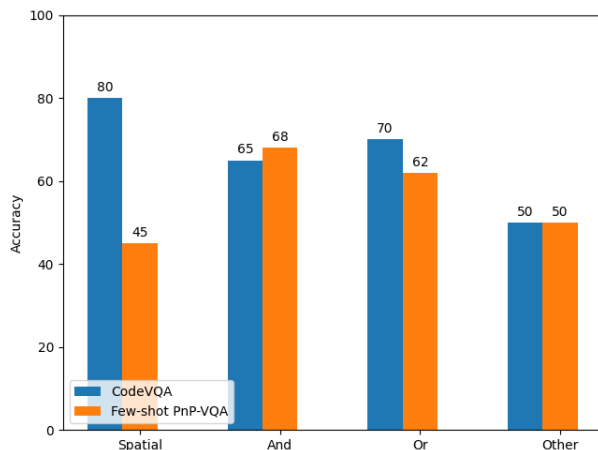


Figure E.1: **Accuracy by question type in 2000 GQA validation examples.** CodeVQA (blue) outperforms Few-shot PnP-VQA (orange) on the spatial and or questions. “Spatial” refers to questions focusing on left/right or top/bottom relations or object positions.




	Few-shot PnP-VQA	CodeVQA (Ours)
 <p>Question: <i>Is the pot to the left or right of the woman wearing glasses?</i> True Answer: Left</p>	<p>Captions: <i>'a bunch of women in aprons cooking in a kitchen', 'three women in hats and a red hat', 'women preparing to cook in kitchen cooking dishes in kitchen', 'female students learning how to cook in a home kitchen', ...</i></p> <p>Answer: Right</p>	<p>Code:</p> <pre>img = open_image("Image13.jpg") woman_pos_x, woman_pos_y = get_pos(img, "woman") pot_pos_x, pot_pos_y = get_pos(img, "pot") if pot_pos_x < woman_pos_x: answer = "left" else: answer = "right"</pre> <p>Answer: Left</p>
 <p>Question: <i>Is the woman on the top part of the picture?</i> True Answer: No</p>	<p>Captions: <i>'people walk by a building and some people walking around', 'a tall sign with people talking to it outside', 'the sign for the hotel has a red - and - white design', 'a person walking next to a tall pole with a sign on it next to a' ...</i></p> <p>Answer: Yes</p>	<p>Code:</p> <pre>LEFT, RIGHT, BOTTOM, TOP = 0, 24, 0, 24 img = open_image("Image13.jpg") woman_pos_x, woman_pos_y = get_pos(img, "woman") if woman_pos_y < (BOTTOM+TOP)/2: answer = "no" else: answer = "yes"</pre> <p>Answer: No</p>
 <p>Question: <i>Do you see both an apple and a sandwich?</i> True Answer: No</p>	<p>Captions: <i>'an apple and two cups of water on a table', 'an apple, a glass lime plate, sitting on a green green table plate', 'an apple, two cups, and a paper plate sits on a table', 'a big red apple on a paper plate', 'an apple is perched beside two plates of food!...</i></p> <p>Answer: Yes</p>	<p>Code:</p> <pre>img = open_image("Image13.jpg") apple_exists = query(img, "Do you see an apple?") sandwich_exists = query(img, "Do you see a sandwich?") if apple_exists == "yes" and sandwich_exists == "yes": answer = "yes" else: answer = "no"</pre> <p>Answer: No</p>

Figure E.2: **GQA Results.** We show example results from the GQA dataset where our method CodeVQA outperforms the baseline Few-Shot PnP-VQA.

E.4 Qualitative Comparisons

We include qualitative comparisons of our method CodeVQA to the baseline Few-shot PnP-VQA (text-davinci-003) in Fig E.2. In all the instances, we can see that PnP-VQA produces captions that are irrelevant to the question, resulting in incorrect answers. On the other hand, CodeVQA breaks down the question into a Python code block. CodeVQA uses if-else conditions along with the pre-defined visual modules `get_pos(image, text)` and `query(image, text)` to focus on the right regions of the image, arriving at the correct answer in an explainable fashion.

Fig. E.3 shows two examples from the NLVR-2 dataset where our method CodeVQA answers the questions correctly. In the first example, it queries each of the images for the count of the pandas, and answers the question correctly based on that. In the second example, our method breaks the question down into three simpler queries and an if-else statement to arrive at the correct answer.



Question: Is it true that there are four pandas?

Code:

```
panda_count = 0
for image in images:
    panda_count += int(query(image, "How many pandas are there?"))
if panda_count == 4:
    answer = "yes"
else:
    answer = "no"
```

Answer: Yes



Question: Is it true that the left image shows laptops in horizontal rows of three and includes rows of open laptops and rows of closed laptops?

Code:

```
images = open_images("ImageSet7.jpg")
rows_of_three = query(images[0], "Are the laptops in horizontal rows of three?") == "yes"
open_laptops = query(images[0], "Are there rows of open laptops?") == "yes"
closed_laptops = query(images[0], "Are there rows of closed laptops?") == "yes"
if rows_of_three and open_laptops and closed_laptops:
    answer = "yes"
else:
    answer = "no"
```

Answer: No

Figure E.3: **NLVR2 Results.** We show example results from the NLVR-2 dataset of our method CodeVQA.

Fig. E.4 shows the correct results of our method on complex multireference questions in the COVR dataset. CodeVQA is able to break down the logic to obtain the counts of images with a cake on a white plate and images with a lemon on a white plate and then evaluates if the two counts are the same.

In the second more complex example, our method uses for loops and complex if-else logic

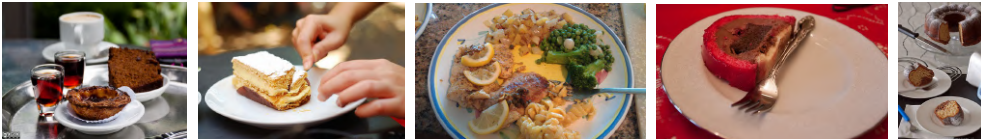


Question: What amount of pictures show girls wearing a skirt and holding a racket?

Code:

```
images = open_images("ImageSet7.jpg")
count = 0
for image in images:
    girl_exists = query(image, "Is there a girl wearing a skirt?")
    if girl_exists == "yes":
        holding_racket = query(image, "Is there a girl holding a racket?")
        if holding_racket == "yes":
            count += 1
answer = count
```

Answer: 1



Question: Are there the same number of images that have a cake on a white plate as there are images that have a lemon on a white plate?

Code:

```
images = open_images("ImageSet7.jpg")
cake_count = 0
lemon_count = 0
for image in images:
    cake_exists = query(image, "Is there a cake on a white plate?")
    lemon_exists = query(image, "Is there a lemon on a white plate?")
    if cake_exists == "yes":
        cake_count += 1
    if lemon_exists == "yes":
        lemon_count += 1
if cake_count == lemon_count:
    answer = "yes"
else:
    answer = "no"
```

Answer: No

Figure E.4: **COVR Results.** We show results on the COVR dataset where our method correctly answers the question by referencing all the images.

to first locate the images that satisfy the criterion, “pillows on a couch near a table” and “pillows on a couch near a bed” to count the individual occurrences.

E.5 Additional Quantitative Results

Table E.1 shows results on validation sets and compares the accuracies of CodeVQA and Few-shot PnP-VQA when using `code-davinci-002` and `text-davinci-003` as the question-answering LM.

Table E.2 shows how the accuracies of CodeVQA and Few-shot PnP-VQA vary with the number of shots in the prompt. Figure E.1 shows the breakdown of accuracy by question type for 2000

Method	Number of shots		
	8	12	16
<i>text-davinci-003</i>			
Few-shot PnP-VQA	48.3	49.4	49.5
CodeVQA	52.8	52.5	52.7
<i>code-davinci-002</i>			
Few-shot PnP-VQA	50.6	52.1	51.2
CodeVQA	55.1	55.3	55.4

Table E.2: Accuracy with different numbers of shots on 2000 GQA validation examples.

GQA validation examples, which we used for initial experimentation (similar to Figure 6.3 but on validation examples). We note that on this sample, Few-shot PnP-VQA has an advantage on “and” questions.

E.6 Experiments with Additional Primitives

We also experiment with two other primitives, on datasets involving counting objects or knowledge retrieval:

find_object(image, object_description) This function returns a set of references to objects in the image that match the given description, and we use it for counting objects. We implement this function using Grounding DINO liu2023grounding, which is an open-vocabulary object detector that is also trained on referring expression comprehension.

We evaluate this primitive on the VQAv2 dataset `vqav2`, for which we use only this primitive and query, as well as the COVR and NLVR2 datasets. We used 12 in-context examples for the VQAv2 dataset. Table E.3 shows the results indicating that using this module for counting rather than query yields mixed results. Qualitatively, we observe a few reasons for errors in the `find_object` version.

First, the object detector is not always accurate (e.g. finding “person holding a banana” when there is a person but no banana). Second, our program may omit key details from the question (e.g. for “How many boats have people in them?” the program counts the number of boats overall). Third, our program may invoke the detector when it ill-suited to the question (e.g. “How many blades of grass surround the fire hydrant?”). On the other hand, captions often convey the number of objects when the number is small, which is very common in these datasets, so query can be effective on counting.

knowledge_query(question) This function returns the answer to a question based on world knowledge (e.g. “Which football team has won the most Super Bowls?”). We implement this function using the same LM that is used for query. In order to better match the format of the OK-VQA dataset, we add a large negative bias to the logits of the following tokens to prevent the LM from generating them: hyphens, “to”, and °. This choice was made based on preliminary experiments on the OK-VQA dataset.

We evaluate this primitive on the OK-VQA dataset okvqa, for which we use only this primitive and query. For CodeVQA and Few-shot VQA, we used 7 in-context examples to be consistent with the OK-VQA results of ViperGPT viperGPT. Table E.4 provides the results, showing that for questions involving both visual information and general knowledge, breaking down the questions in this way does not lead to improved accuracy.

For both VQAv2 and OK-VQA, we use the standard evaluation method associated with the VQAv2 dataset, which takes into account the set of ground-truth answers for each question. The Flamingo flamingo results that we report on both datasets used 32 in-context examples.

	VQAv2	COVR	NLVR2
Zero-shot			
BLIP-v2	65.0	–	–
Few-shot			
Flamingo	67.6‡	–	–
Few-shot PnP-VQA	66.84	47.8	63.4
CodeVQA	–	52.9	64.0
CodeVQA w/ find_object	66.63	52.9	66.0

Table E.3: Results with find_object used for counting objects on VQAv2 (a random sample of 4000 examples from validation set), COVR (validation), and NLVR2 (test-public). ‡indicates a result on the full VQAv2 test-dev set, which may not be directly comparable with our results on a sample of the validation set.

	OK-VQA
Zero-shot	
BLIP-v2	45.9
Few-shot	
Flamingo	57.8
ViperGPT	51.9
Few-shot PnP-VQA	54.1
CodeVQA	53.5
w/ knowledge_query	

Table E.4: Results with knowledge_query on the OK-VQA validation set.

E.7 Licenses and Other Dataset Details

GQA is licensed under the CC-BY-4.0 license (<https://creativecommons.org/licenses/by/4.0/>). The COVR repository (<https://github.com/benbogin/covr-dataset>) is licensed under an MIT license (though imSitu images may not be licensed). The text in both datasets is written in English. The annotations in NLVR2 are licensed under CC-BY-4.0, but the images in the dataset are not licensed. The annotations in VQAv2 are licensed under CC-BY-4.0.

The testdev set of GQA contains 12578 instances. The test set of COVR contains 7024 instances. The validation set of COVR contains 6891 instances. The public test set of NLVR2 contains 6967 instances. The validation set of OK-VQA contains 5046 instances. For VQAv2, we evaluate on a random sample of 4000 examples from the validation set.

During the development and intermediate evaluations of our method, we evaluated on a random sample of 200 training examples and a random sample of 2000 validation examples from GQA, a random sample of 200 training examples and the validation set from COVR, a random sample of 2000 training examples from NLVR2, a random sample of 1200 training examples from OK-VQA, and a random sample of 2200 training examples from VQAv2.